MEDINA

Deliverable D4.2

Tools and Techniques for the Management and Evaluation of Cloud Security Certifications – v2

Editor(s):	Immanuel Kunz
Responsible Partner:	Fraunhofer Institute for Applied and Integrated Security AISEC
Status-Version:	Final – v1.0
Date:	31.10.2022
Distribution level (CO, PU):	PU

Project Number:	952633
Project Title:	MEDINA

Title of Deliverable:	Tools and Techniques for the Management and Evaluation of Cloud Security Certifications – v2
Due Date of Delivery to the EC	31.10.2022

Work package responsible for the Deliverable:	WP4 – Continuous Life-Cycle Management of Cloud Security Certifications
Editor(s):	Immanuel Kunz (FhG)
Contributor(s):	AISEC, XLAB, TECNALIA, NIXU, Bosch
Reviewer(s):	Mirko Manea / Debora Benedetto / Claudia Zago (HPE) Cristina Martínez (TECNALIA)
Approved by:	All partners
Recommended/mandatory readers:	WP3, WP5, WP6

Abstract:	This deliverable contains contributions towards the automation of certification evaluation and management steps, as well as risk assessments and possible mitigations regarding the protection of evidence and certificate management. It is the result of work on the tasks T4.1, T4.2, and T4.3 up until month 24 of the project. It is the second version of the first WP4 deliverable and will be followed by a further iteration (D4.3).
Keyword List:	Certificate Evaluation, Certificate Management, Distributed Ledger Technologies, Smart Contracts
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	11.07.2022	Initial TOC	FhG
v0.2	30.09.2022	First draft version	FhG, TECNALIA, XLAB
v0.3	18.10.2022	Updated draft which includes all results from D4.1	FhG
v0.4	19.10.2022	Version ready for internal review	FhG, TECNALIA, XLAB
v0.5	26.10.2022	Addresses all comments received in the internal review	FhG, TECNALIA
v1.0	31.10.2022	Ready for submission	TECNALIA

Table of Contents

List of Tables

List of Figures

Terms and abbreviations

API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CAB	Conformity Assessment Body
CCE	Continuous Certification Evaluation
CIA	Confidentiality, Integrity, and Availability
CFT	Crash Fault Tolerance
CSC	Cloud Service Customer
CSP	Cloud Service Provider
DID	Decentralized Identifiers
DLT	Distributed Ledger Technologies
DoS	Denial of Service
EBSI	European Blockchain Services Infrastructure
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
GPL	General Public License
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
IPFS	Interplanetary File System
KPI	Key Performance Indicator
HW	Hardware
IaaS	Infrastructure as a Service
IBFT	Istanbul Byzantine Fault Tolerance
IoT	Internet of Things
LCM	Life-Cycle Manager
LGPL	Lesser General Public License
MitM	Man in the Middle
MIP	Moving Intervals Process
NIST	National Institute of Standards and Technology
NSA	National Security Agency
PaaS	Platform as a Service
PBFT	Practical Byzantine Fault Tolerance
PoET	Proof of Elapsed Time
PoA	Proof of Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
QHP	Quantitative Hierarchy Process
QPT	Quantitative Policy Trees
QR	Quick Response
SaaS	Software as a Service
SHA	Secure Hash Algorithm
SLA	Service Level Agreement
SLO	Service Level Objective
secSLA	Secure Service Level Agreement
SPoF	Single Point of Failure
SSI	Self-Sovereign Identities
SSL	Secure Sockets Layer
SW	Software

TOC	Target Of Certification
ToE	Target of Evaluation
TOM	Technical and Organizational Measure
TPS	Transactions Per Second
TSL	Transport Layer Security
UI	User Interface
VPN	Virtual Protocol Network
WP	Work Package
ZKP	Zero-Knowledge Proof

Executive Summary

This document presents the second iteration of *tools and techniques for the management and evaluation of cloud security certifications* developed in WP4.

Evaluating, managing, and protecting certificates and their underlying evidence are challenging tasks, which do, however, have potential for automation. This deliverable targets these challenges in three parts.

First, an approach and technical prototype for the evaluation of assessment results is presented (*Section 4*). Assessment results are created in WP3 and are forwarded to this *Continuous Certification Evaluation* component. Here, they are aggregated in a continuous manner, building a tree structure of the certification that is both machine-readable and quickly understandable by humans, e.g., auditors.

Second, an approach to securing the integrity of evidence and assessment results is identified (*Section 5*). This includes a risk analysis for the evidence and assessment results that underly every certificate's state changes and the evaluation of possible mitigations, including Blockchain and many Blockchain-like technologies that have emerged recently.

Third, an approach and implementation for a certificate *Life-Cycle Manager* is presented (*Section 6*). It implements a state machine that reflects the EUCS-defined certificate states. Its approach includes an evaluation of different technological approaches to implementing and securing certificates, including smart contracts and self-sovereign identities, and a discussion about the different approaches to automating certificate state changes.

We describe background concepts and related work in *Section 2* and present the overall architecture of the WP4 components in *Section 3*. This deliverable is related to WP3, since the WP4 components directly process the results of the components developed in WP3. The overall purpose and integration of the components described here in the MEDINA framework is furthermore described in WP5.

This deliverable presents the second version of the WP4 components – except of the Risk Assessment component which is described in the deliverable D4.4 [1] (and its future iteration D4.5). One more version of the deliverable at hand (D4.3) will be created in the future. The following additions are planned for this future iteration:

- Final integration: All newly developed features will be integrated with other components. Especially using the *target of evaluation* entity (see Section 2.3), allowing to support multiple cloud services and certification frameworks.
- Improvement of the certificate report workflow: Currently, simple reports of suspended and withdrawn certificates are reported, but no responses are processed.
- Improved integration of operational effectiveness metrics, e.g., in the UI.
- Analysis of Blockchain technologies' energy consumption, cost, and scalability.
- Improvement of the SSI-Framework focusing on usability and security aspects, including Zero-Knowledge Proofs (ZKPs) support.

1 Introduction

In MEDINA we present several innovations that together enable the continuous certification of cloud services. These include a common abstraction and language for requirements and metrics including an ontology (see D2.1 [2]), as well as the continuous gathering and assessment of evidence (see D3.2 [3]). In WP4, we address the final aggregation and evaluation of the assessment results to crystallize a concrete decision about a certificate's state.

Normally, manual audits are conducted by external auditors to verify a set of pre-defined requirements at a point in time. Consequently, the decision of issuing a certificate is made by humans considering various sources of evidence, like documentation, data samples, and interviews. Auditors will therefore not only evaluate specific pieces of evidence, but also the gathered evidence as a whole. This process allows for some amount of consideration by the auditors who can evaluate the fulfilment of requirements depending on the context.

Translating this process to a technical implementation has advantages and disadvantages: On the one hand, an automated certification process can provide continuous auditing, improved traceability of decisions, and a more standardized, comparable process. On the other hand, an automated certification process has a rigid focus on evidence that can be gathered technically, e.g., configurations of cloud resources, and on the set of metrics that is available. Consequently, while human auditors may weigh different kinds of evidence considering the service's context, an automated implementation needs clearly defined requirements and decision criteria independently of the service's context.

Furthermore, certificates need to be managed. In the traditional certification process, where audits are not done continuously, a certificate can simply be published, for instance in a public registry. In the continuous model, however, any newly gathered evidence can have a major impact on the certificate's state. Its state therefore needs to be evaluated continuously as well, and its publication needs to be managed which harbours risks if done automatically.

Finally, the results generated by such components, as well as the components' logic, also need to be protected against intentional and unintentional threats. For instance, certificate state changes need to be verifiable to establish trust into this automated process.

1.1 About this deliverable

The deliverable at hand describes MEDINA's contribution towards the continuous evaluation of security assessments of cloud services. These contributions include an approach for continuously aggregating assessment results, as well as deriving a decision about the certificate state.

The proposed approach for continuous aggregation of assessment results represents a certification as a tree-like structure that is evaluated based on its leaves—the assessment results. Thereafter, the risk assessment component processes the results qualitatively to consider service-specific criteria, like especially impactful resources (described in D4.4 [1]). Finally, the life-cycle management component reflects a certificate's state as a traceable state machine. It makes state change decisions based on deviation reports by the risk assessment. To protect its correct execution, different technologies, like smart contracts, are considered and evaluated.

The deliverable furthermore includes the research results for the protection of evidence and assessment results, whose implementation is described in D3.2 [3]. This includes the identification of risks for these artefacts, and an evaluation of possible mitigative technologies.

These contributions aim at yielding the advantages mentioned above, e.g., an improved traceability and automation, while at the same time addressing potential risks and challenges of managing and protecting certificates.

1.2 Document structure

The deliverable is structured as follows. Section 2 describes background concepts and other work related to the following sections. In Section 3, the overall architecture and goals of the developed components are described. Thereafter, Section 4 presents the MEDINA approach to evaluate assessment results, aggregating them into a certification tree. Next, the risks that evidences and assessment results are exposed to are analysed in Section 5, laying the groundwork for the implementation of the *MEDINA Evidence Trustworthiness Management System*. Section 6 then presents the approach and current status of the life-cycle management and the *SSI Framework*. Section 7 concludes the deliverable.

1.3 Updates from D4.1

This deliverable presents the second iteration of work done in WP4. Please note that some sections (especially parts of 4.1.3, and Sections 5.1 and 6.1) have not changed from D4.1 [4]. The following list summarizes the main changes:

- The Background and Related Work section has been extended (Section 2).
- A new data entity, the *Target of Evaluation*, has been introduced (Section 4).
- The concept of *Operational Effectiveness* has been introduced into the certificate maintenance. It indicates the compliance of a certificate over time (Section 4.1.3 and Section 6.2.4).
- For the audit trail, an analysis of hashing techniques has been added (Section 5.3).
- Different alternatives for the verification of evidence and assessment results stored in the *MEDINA Evidence Trustworthiness Management System* have been developed (Section 5.4).
- The *Life-Cycle Manager* (*LCM*) now includes automatic suspension and withdrawal rules according to rules defined in the EUCS. It also implements a report of suspension or withdrawal of certificates to the *SSI Framework* (Section 6.2).
- A proof-of-concept implementation of the *SSI Framework* has been finished (Section 6.3).
- The Continuous Certification Evaluation component has been extended with a number of additional features, support for connectivity with other MEDINA components, UI, and ability to calculate operational effectiveness metrics (Section 4).

2 Background and Related Work

This section explains background concepts and related literature that build the foundation for the work described in the rest of the deliverable.

2.1 Evaluating Cloud Security Certifications

Evaluation of security compliance in MEDINA starts with the gathering of evidence in WP3 components. Security assessment components assess this evidence based on the target values as configured for the specific requirement and provide their output (assessment results with the state of fulfilment of a specific metric for a specific monitored resource) to the *Continuous Certification Evaluation* component. If the assessment result value represents the lowest-level information about the certification state, the role of the *Continuous Certification Evaluation* component is to combine the received assessment results into information about the fulfilment of higher-level certification objects: requirements, controls, control groups, and the selected certificate scheme in its entirety. This information does not directly determine the cloud service's eligibility for a certificate, but serves as input for other components, the *Risk Assessment and Optimisation Framework* (described in D4.4 [1]) and the Certificate Lifecycle Management (see Sections 2.5 and 6), as well as for an easy visualisation of the certificate state for the users (CSPs and auditors).

To assist the design of the *Continuous Certification Evaluation* component, previous research about similar problems was consulted. Luna et al. [5] presented two methods (based on Quantitative Policy Trees (QPT) [6] and Quantitative Hierarchy Process (QHP) [7]) for quantitatively assessing whether (and to what extent) a CSP fulfils the security requirements expressed by a customer, and the general level of security offered by a CSP. Their method is based on cloud security Service Level Agreements (secSLAs), which consist of various Service Level Objectives (SLOs) that map to one or more measurable metrics. Cloud Service Customers (CSCs) express their security needs by defining thresholds for the values of metrics and weights (importance) of the individual SLOs (QPT) or all levels of the SLA hierarchy (QHP). The CSPs' secSLAs are evaluated with respect to the customer's security requirements to output a ranking of CSPs according to their level of fulfilment of these requirements.

Modic et al. [8] improved the computational efficiency of the previously presented QHP method and developed a high-performance technique, Moving Intervals Process (MIP), which, beside checking the fulfilment and potential under-provisioning of CSC's requirements, also rates CSPs based on how much the customer's requests can be over-provisioned by the cloud service. Like QHP, MIP also uses calculations based on weighted arithmetic mean to aggregate values of SLOs to all levels of the secSLA hierarchy. According to the level of fulfilment of a customer's requirement, MIP assigns values to SLOs on the interval [0,2] where values less than 1 represent under-provisioning, 1 is assigned where the CSP exactly meets the requirement, and values greater than 1 represent over-provisioning. Because some of the values on the same hierarchy level can be greater than 1 and others less, the aggregated value can result in apparent over-provisioning (>1) even though some child values do not even meet the customer's requirement. The authors suggested a correction to the scores to eliminate this masquerading effect. Both of the mentioned methodologies for cloud security evaluation were (developed and) used in the EU FP7 project SPECS [9].

Maroc and Zhang [10] proposed a cloud security evaluation approach that additionally features a risk-driven selection of evaluation criteria and considers multiple factors in weighting of criteria: user's preferences, criteria interdependencies, the type of cloud service (IaaS / PaaS / SaaS) that determines the user's level of control, as well as relations between threats and vulnerabilities, their risk (likelihood and impact), and security controls.

In MEDINA, the *Continuous Certification Evaluation* component does not give the final evaluation of the security and certificate state, but its output is combined with a separate risk assessment framework that considers values of assets and their potential risks. For this reason, the *Continuous Certification Evaluation* does not deal with the additional risk-driven parameters as proposed in [10], but focuses on effectively aggregating the information received by assessment results.

As mentioned, the problem addressed in [5] and [8] was to rank CSPs according to the CSC's needs. The problem addressed in MEDINA is slightly different, as here the CSP's compliance is determined with a specific standardization (level). In the typical case, all controls, and requirements of a standard need to be fulfilled for the cloud service to be (or to remain) certified, although a minor non-conformity occurring for a limited amount of time does not invalidate the certificate. For this reason, it can be useful (for user's review as well as further risk calculation) to observe the level of fulfilment at all layers in the standard's hierarchy, not only the binary information about (non-)conformity.

The methodology used in the *Continuous Certification Evaluation* component is thus based on building the evaluation tree with assessment results in its leaves, aggregated according to the standard's hierarchy. The aggregation is done with weighted arithmetic means, following the approaches mentioned above. The approach from [8] can be simplified though as assessment results in MEDINA only include binary values (1 meaning conformity and 0 meaning inconformity), which means that there is no over-provisioning, and the masquerading effect does not apply. Additionally, since the goal is to also present intermediate fulfilment values in all levels of the aggregation tree (not only at its root for the entire certification fulfilment), thresholds should be set to determine the fulfilment in individual tree nodes (controls, control groups, etc.). These thresholds and the aggregation weights of the nodes can be set by the user or the auditor (e.g., based on the importance of evaluated resources or controls). The evaluation tree can be easily simplified to an AND tree by setting the thresholds in all nodes to 1, meaning that all the assessment results must indicate fulfilment for the evaluation to be positive, irrespective of the assigned weights (as long as they are positive). The design of the *Continuous Certification Evaluation* component is further explained in Section 4.

2.2 Operational Effectiveness

Beside the calculation of the current state of the evaluation tree nodes, the *Continuous Certification Evaluation* component also provides information about the evaluation history supported by metrics of operational effectiveness. These are metrics that measure, in various ways, how well a particular requirement or control was established (fulfilled) in a certain period of time. If a control is unfulfilled for a small amount of time, this is typically not a big issue for the entire certificate state. On the other hand, if the problem has not been mitigated for a long time, the certificate may be revoked.

The amount of time that the CSP needs to correct the issue in question or how often the control is non-compliant are examples of operational effectiveness measures that can be important for evaluating the overall certification eligibility. Stephanow and Banse [11] introduce four universal metrics for continuous test-based certification evaluation techniques. The metrics discussed are: Basic-Result-Counter (counting the number of passes and fails for a test), Fail-Pass-Sequence-Counter (a fail-pass sequence meaning one or several consecutive test fails followed by at least one pass), Fail-Pass-Sequence-Duration (measuring the time between a first failed test in a sequence and the next passed test), and Cumulative-Fail-Pass-Seq-Duration (returning the sum of all fail-pass sequences' durations).

2.3 Target of Evaluation

The Target of Evaluation (ToE) binds a cloud service to a certification framework (or catalogue). The introduction of this concept was necessary to support multiple cloud services and certification frameworks: previously only one service could be monitored and certified, while now there is an *n:m* relation between the two. A ToE is created and managed in the *Orchestrator* and then propagated to the other components (please note that the implementation of this feature is still in progress at the time of writing).

2.4 Digital Audit Trails

MEDINA framework includes digital audit trails as security mechanisms to improve the integrity, traceability and availability of the most relevant information considered in MEDINA (evidence and assessment results). Digital audit trails are detailed and chronological records of important information that are usually used to verify and track all related processes (updates).

Nowadays, audit logs provide a useful service, allowing auditing processes, secure information storage, tracking of changes made to recorded data (audit trail) and discrepancies, anomalies, and malicious activities detection. However, current audit logs implementations can be vulnerable to different types of attacks, which enable adversaries to tamper data and audit logs. Thus, integrity could be compromised. In addition, audit logs are usually under the control of a central authority which controls and manages information records.

To counter the aforementioned attacks, Blockchain technology has started to be considered as a technology for auditing purposes. One promise that Blockchain technology makes is to move trust from a central authority to a distributed network. Also, Blockchain creates an immutable record of transactions, so an immutable audit data storage that is not governed by a central authority can be provided.

In general terms, Blockchain is a Distributed Ledger Technology (DLT) created over a distributed and decentralized network of peer nodes which maintain a copy of the ledger by applying transactions that have been previously validated by a consensus protocol and grouped into blocks with a cryptographic hash that binds each block to the preceding block. This way, given the last block, the previous ones cannot be modified without altering subsequent blocks (i.e., data is practically resistant to modification). Another key aspect of these data structures is that transactions are digitally signed so the origin of a piece of data can always be traced back to its creator. Additionally, Blockchain eliminates the need for a central control authority to manage transactions or keep records. The main features for Blockchain technology are:

- Decentralization: There is no central authority and no central data storage
- Trustlessness: Blockchain does not require trust in a central authority or any single participant
- Transparency and traceability: All transactions in a Blockchain are visible and verifiable
- Immutability: Transactions and blocks added to the Blockchain are practically impossible to manipulate.

These features are beneficial for audit trail systems like the one implemented in the MEDINA framework.

2.5 The Cloud Security Certification Life-Cycle

Increasing the degree of automation in the management of certificates requires first modelling and then implementing the possible certificate states. The EUCS [12] defines several such states: *Renewed*, *Continued*, *Updated*, *New Certificate*, *Withdrawn*, and *Suspended*. An issuance or state change follows a review by the CAB. In the literature, different (semi-)automated life-cycle

models can be found for cloud security certifications, defining different states and state change procedures.

Cimato et al. [13] first proposed a complete certification model for cloud systems, addressing the problem of certifying a dynamically provisioned system in a continuous way. They develop a meta-model with different modules, including a certificate module which proposes a simple certificate life-cycle as shown in Figure 1, that includes the states *Valid*, *Revoked*, *Renewed*, and *Invalid*. They do not, however, detail how the certification transitions are decided [14].

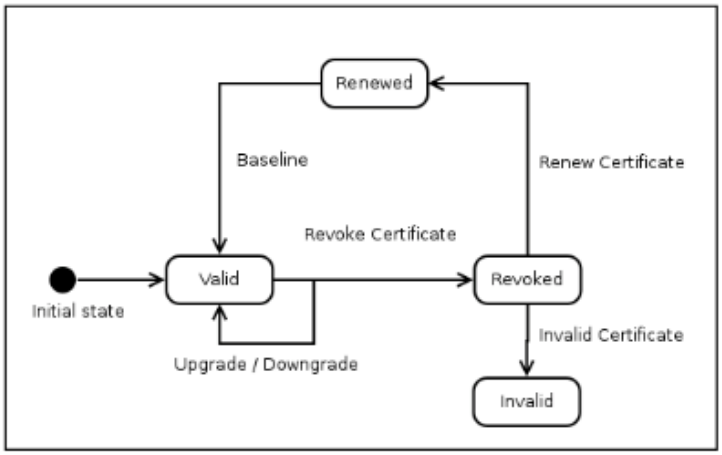

Figure 1. Certificate Life Cycle proposed by Cimato et al. [13]

A further proposal by Ardagna et al. [14] describes a certification process that comprises the phases *Monitor*, *Analyse*, *Plan*, and *Execute*. After one iteration of these phases, the process generates a certificate along with supporting evidence. Further evidence is then collected continuously to verify the validity of the certificate. Please note that non-compliances are corrected automatically to ensure the validity of the certificate. This model therefore goes beyond the focus of MEDINA which does not enforce certification requirements, but aims at collecting various evidence from different sources, aggregating them, and deriving a sophisticated certification decision.

Kunz and Stephanow [15] define a process model for the continuous certification of cloud services based on two main requirements. First, the target of certification (TOC) may change frequently, so a frequent re-discovery of the TOC needs to be done. This requirement is addressed in MEDINA in WP3. Second, the certificate's state may change any time based on the results of the certification techniques and needs to be reported. They also discuss the implications of automatically reporting certificate updates. Furthermore, they note that several degrees of automation can be targeted in between the traditional, manual process, and the completely automated one. They define three high-level phases for the traditional, manual process which are derived from several certification standards: *Initialization*, *Audit*, and *Certification*, which are repeated in cycles. Their proposal adds a *Scoping* phase to define the scope of the service to be audited which includes the discovery of existing cloud resources.

Anisetti et al. [16] propose a semi-automated certification scheme that includes the following phases: *Not Issued*, *Issued*, *Suspended*, *Expired*, and *Revoked*. They furthermore define transition conditions as shown in Figure 2, which presents a finite state automation. Existing approaches of (semi-)automated certification usually start with an initialisation phase that sets up the necessary tooling, e.g., discovery mechanisms, smart contracts, etc., and aim at verifying the certificate's current state thereafter automatically—or changing it if it doesn't comply with the pre-defined conditions.

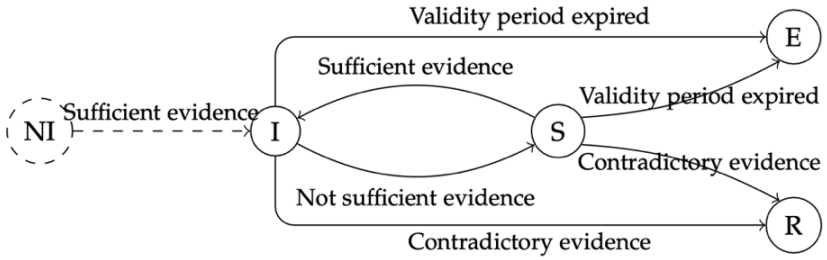

Figure 2. Certificate State-Change Model by Anisetti et al. [16]

In this deliverable, the state model for security certificates is based on the states and change criteria defined in the EUCS [12] and is implemented in a finite state automaton similar to the one described in [16].

Proposals in other EU projects: AssureMOSS

The AssureMOSS[1] (Assurance and certification in secure Multi-party Open Software and Services) project develops methods and tools that support the secure development of open software throughout its entire life-cycle. Work package 5 of AssureMOSS includes the development of a risk assessment method as well as the automatic re-certification based on the risk assessment results [17]. In the certification model and certification life-cycle proposed, there are overlaps with the proposals described in Section 6.2.2. For example, the AssureMOSS methods aim at covering different certification frameworks, including the EUCS, and differentiate between an initial "baseline certification" and following "delta certifications", where the initial audit is confirmed by a human auditor and can be confirmed automatically thereafter. This confirms the assumptions made in MEDINA: The configuration of tools, as well as certain parameters and thresholds need to be acknowledged by an auditor before they can be trusted. Also, the authors differentiate between "minor changes" and "major changes" in the system which conform to the minor and major deviations defined in the EUCS and used in this deliverable as well. Regarding the certification life-cycle, the project proposes a *valid* state, and different *invalid* states (expired, revoked, obsolete) where the validity may decrease over time. In this regard, MEDINA is more focused on the EUCS-defined states. The most challenging point in both proposals, however, is how certificate transition decisions should be made. Here, the deliverable at hand includes more sources of information than AssureMOSS, e.g., including the operational effectiveness measures.

[1] https://assuremoss.eu

3 Architecture

This section describes the overall architecture of the WP4 components. First, the overall goals are explained. Then, the architecture is presented and described. More detailed descriptions of the components and data models are included in the following sections.

3.1 Design Goals

Overall, the goal of WP4 is to process the gathered, and pre-assessed evidence and consequently decide on the certificate state. To that end, several steps are necessary. First, the assessment results need to be aggregated according to their certification requirements. This step needs to be executed continuously since assessment results are generated continuously by the WP3 components as well. Second, the result of this aggregation needs to be enriched using service-specific information. This step is necessary because not all non-compliances are equally severe. Only after this step has been done, an informed decision on the existence of significant deviations can be made, and a translation to a state change can be done.

The components therefore need to process data, like assessment results, continuously. They should also be independently executable to allow for different deployment options. The *Life-Cycle Manager*, for example, may be deployed by the CSP to manage different certificates. However, it can also be used by certification body to manage the state of their customers' certificates. While this work package aims at automating large parts of the certification evaluation process, the developed components should also present a useful means for internal and external auditors, for instance to investigate deviations.

3.2 Architecture Overview and Data Flow Model

Figure 3 shows an overview of the developed architecture which is described in the following. The entry point of the WP4 components is the interface between the *Orchestrator* (WP3) and the *Continuous Certification Evaluation* component (CCE). This data flow is designed as a stream of assessment results that are sent to the CCE (see Section 4). The CCE in turn aggregates the assessment results to evaluate the overall certificate status on different levels of its hierarchies, e.g., its requirements and controls. The result of this evaluation is an impact-agnostic, tree-based representation of the certificate's compliance state. Only in the next component, the risk assessment, are the results evaluated in more detail considering their individual context, possibly including threat and impact levels. This detailed risk assessment allows then to make an informed decision about the certificate state in the *Life-Cycle Manager*, which reports it to the *SSI Framework* (see Section 6).

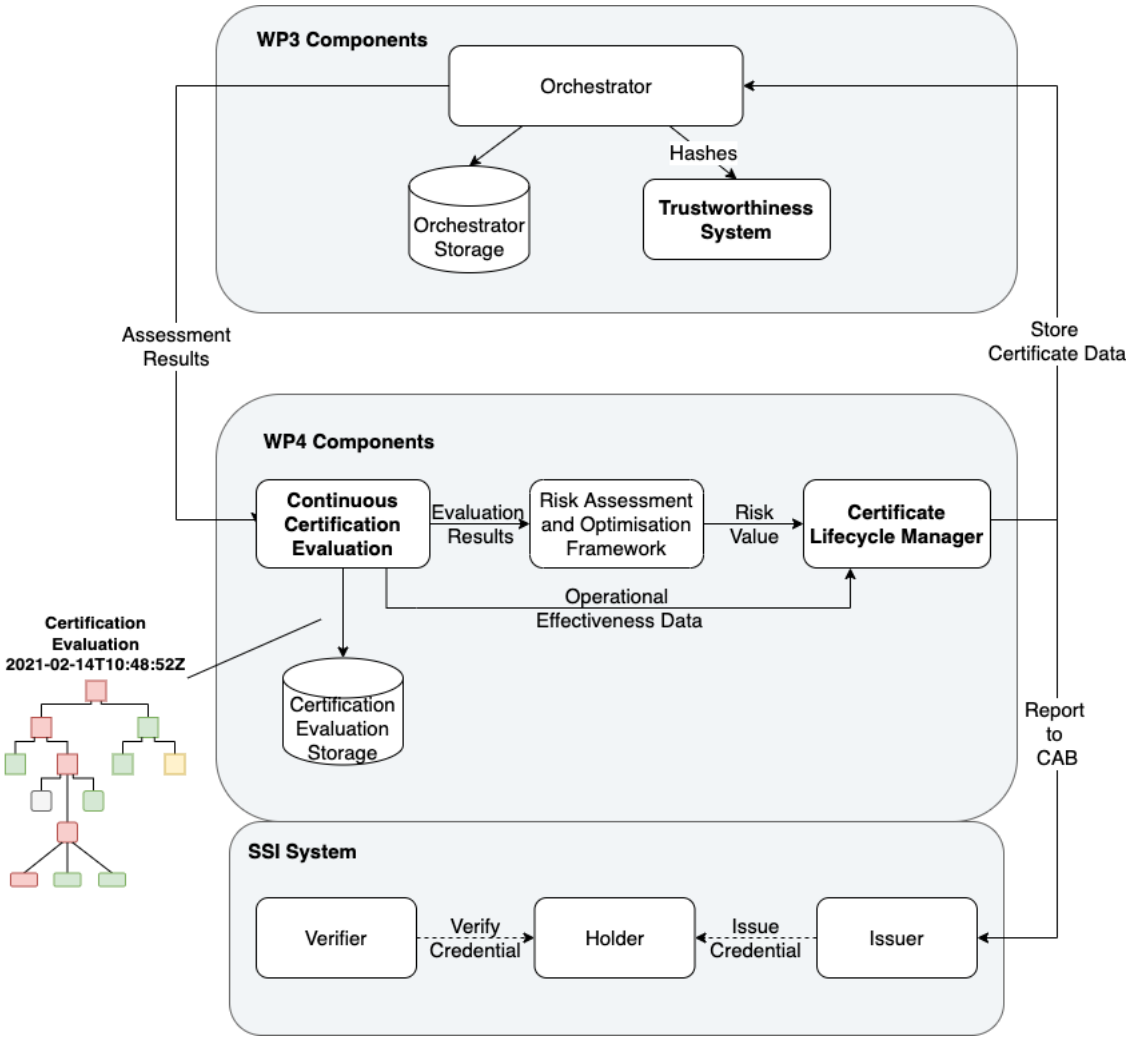

Figure 3. Overall Architecture of WP4 components and connection to WP3 components (source: MEDINA's own contribution)

4 MEDINA Continuous Evaluation of Cloud Security Certification

This section describes the *Continuous Certification Evaluation* (CCE) component of MEDINA and the methodology used for its implementation. The following subsections describe the methodology, functionalities, and requirements to be fulfilled by this component, as well as its design and implementation.

4.1 Methodology

As explained in Section 2.1, the method for aggregation of assessment results in the *Continuous Certification Evaluation* follows the tree-like hierarchy of the various standardisation schemes, like shown in Figure 7. Values in the tree are evaluated bottom-up: from the leaves that represent assessment results to the root representing the complete certification scheme and thus indicating the fulfilment of the certificate.

The general design of the Evaluation component is modular and adaptable in terms of aggregation and tree building. The aggregation can be made with various methods, also by combining different methods at different levels of the tree. The tree skeleton can be built in advance if all the relevant resources and their mappings to requirements and metrics are known before gathering the evidence. Alternatively, the tree structure can be built while receiving assessment results and discovering the resources and the requirements that they must fulfil.

The current proposal of the methodology which is closely related to the *Risk Assessment and Optimisation Framework* is described below.

4.1.1 Building the Tree Structure

The evaluation tree (see Figure 7) is logically composed of two parts: in the upper part, the structure, that is defined directly by the scheme being used, i.e., its control groups, controls, and requirements. There is a possibility that controls can be selected or unselected by the user if allowed by the standardisation scheme in use. The levels below the requirements level are not directly defined by the standard but are important for determining the compliance values of the elements higher in the hierarchy. The conformity to a requirement is determined by measuring one or more metrics related to this requirement, and there can be multiple resources on which the measurements are made. A single assessment result contains the information about whether a particular monitored resource conforms to the target value for a specific metric. To use the assessment results for computing the conformity values of requirements, three aggregation techniques are described below:

a) directly aggregating assessment results into compliance values of requirements,
b) combining assessment results of different resources into compliance values of metrics, and combining metrics into compliance values of requirements,
c) combining assessment results of different metrics into compliance values of resources and combining resources into compliance values of requirements.

The above-mentioned techniques are represented graphically in Figure 4. Possible options for aggregation of assessment results into compliance levels of requirements (source: MEDINA's own contribution). Technique a) is the simplest, avoiding the additional aggregation layer. The downside of this approach is the lack of visibility of metrics' or resources' compliance levels – the compliance levels of resources or metrics are not computed and cannot be examined by users. Also, aggregation weights of metrics and resources cannot be assigned individually but must be combined into a single value.

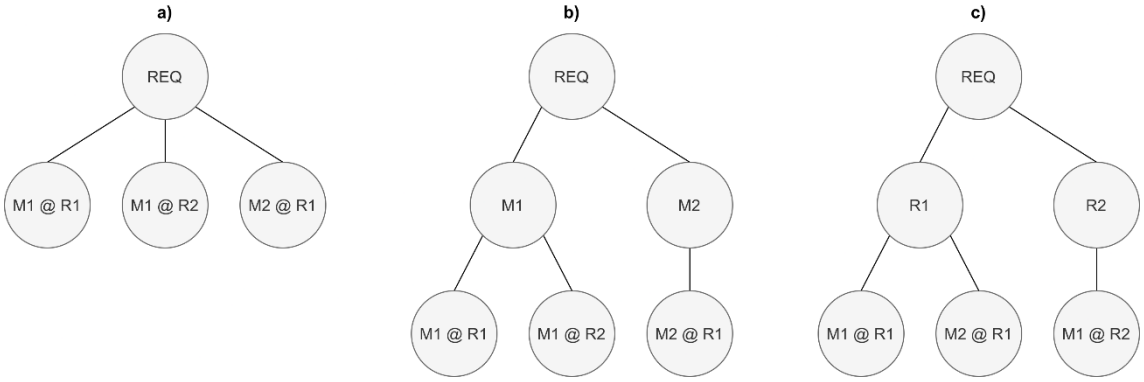

Figure 4. Possible options for aggregation of assessment results into compliance levels of requirements (source: MEDINA's own contribution)

The difference between options b) and c) is whether assessment results are aggregated into compliance values of metrics or into compliance values of resources, respectively. Technique b) calculates whether some metric is satisfied across all relevant resources, whereas option c) evaluates whether some resource is satisfying the related requirement considering all relevant metrics. The aggregated value at the requirement level will be the same in both options b) or c) in case when values of all metrics under the requirement in question have been measured for all relevant resources. When this is not the case (example shown in Figure 4 and described below), the requirement value computed using technique b) is affected greater by non-conformities in assessment results of metrics, measured on fewer resources, while with technique c) assessment results of resources where fewer metrics are measured are regarded as more important (considering similar aggregation weights).

Let us consider the example shown in Figure 4, where both metrics 1 and 2 are evaluated for resource 1, but only metric 1 is evaluated for resource 2 (no assessment result was obtained for metric 2 on resource 2). For this example, we assume that weights for all resources and all metrics are the same. In case all assessment results are positive, the requirement value is 1 for all methods. Table 1 shows a comparison of the calculated fulfilment value for a requirement when one of the assessment results is negative (the other two are assessed as positive) with different methods of aggregation. Method a) considers all three assessment results equally, thus the requirement value is $^2/_3$, regardless of which single assessment result is negative.

When metric 1 at resource 1 is evaluated negatively, both b) and c) methods return the same requirement value since (as evident from Figure 4) this assessment result is represented as half of metric 1 (b) and also as half of resource 1 (c). In case when metric 2 at resource 1 is evaluated negatively, method b) returns a requirement value of 0.5, while it is 0.75 with method c). Method b) penalizes this case harder because this is the single assessment result for metric 2 – one of the two metrics in the second tree level is evaluated with 0. Analogously, when metric 1 for resource 2 is non-conformant, it is penalized harder with method c) since this is the only metric evaluated on resource 2.

The last column in the table shows the c) method of aggregation where the aggregation results for different metrics are aggregated with the AND approach – each resource is assigned a Boolean fulfilment value: 1 if and only if all metrics for this resource are evaluated positively; 0 otherwise. With this method, the requirement values for all cases in our example are 0.5 because one of the two resources on the second tree level is evaluated negatively in each case. Metric aggregation using AND rule is further discussed below.

Table 1. Comparison of requirement fulfilment value depending on non-conformity of individual assessment results calculated with different aggregation methods

	a)	b)	c)	c), AND metric aggregation
M1 @ R1 negative	0.67	0.75	0.75	0.5
M2 @ R1 negative	0.67	0.5	0.75	0.5
M1 @ R2 negative	0.67	0.75	0.5	0.5

If the evaluation tree is built using technique b), the users are able to see the conformity level by metrics and, if needed, they can examine the individual metrics to discover which resources under this metric are contributing to some non-conformity. Alternatively, with option c), users can see the conformity levels of their resources with all metrics linked to a requirement aggregated. They can examine the lowest tree level (metrics) to determine which metrics in that resource are problematic. Due to this, we believe that most users would find option c) more informative.

As explained in Section 4.1.2, the approach for the initial MEDINA proof-of-concept considers that all metrics for a particular resource need to be evaluated positively to regard the requirement fulfilled. Aggregation of the metrics level is thus made with simple AND rules and weighted aggregation does not apply at this level. On the other hand, configuration of different weights for resources can be desirable from the risk assessment perspective. With aggregation technique b), fulfilment values of metrics are calculated from multiple resources and are thus not Boolean values. If we are to apply AND aggregation on metrics, we could consider the metrics values positive or negative depending on thresholds. Regardless of thresholds though, the weights of resources used on the leaf-level would become irrelevant at the requirement and higher levels of the evaluation tree (Boolean fulfilment values are applied to requirements). With technique c), the assessment results are aggregated into resources' compliance levels using AND, applying Boolean values to resources. The compliance values of resources can therefore be aggregated into requirements' fulfilment levels using their respective weights.

Following the considerations described above, technique c) was chosen for the initial implementation of the Continuous Certificate Evaluation component and is therefore considered in the following description of the tree-building process. As an example, a part of such evaluation tree is also shown in Figure 7. As mentioned, the component is implemented in an adaptable way, meaning that if additional requirements are found, refinements of the approach are possible and would not require significant effort.

The first proof-of-concept implementation of the MEDINA framework does not plan to include a repository of all monitored resources related to the evaluated cloud service. For this reason, the entire evaluation tree structure cannot be built in advance (before receiving the assessment results for individual resources). In the start-up phase of the component, the tree structure is built down to the level of requirements by obtaining the elements of the certification scheme and the mappings between the hierarchy levels from the Catalogue of controls and metrics. The lower part of the tree is built part by part during the component's operation.

When receiving an assessment result for metric *M* and resource *R*, the component first checks whether such an assessment result is already present in the evaluation tree. In this case, its values (there can be multiple tree nodes corresponding to an assessment result when a single metric maps to several requirements) can simply be updated and propagated to the higher hierarchy levels through aggregation. If no nodes with metric *M* and resource *R* exist, they need to be added to the tree. Resource *R* is added as a child node to all requirements that metric *M* is associated with. For all such added nodes of resource *R*, metrics that are required for

fulfilment of particular requirements are added as child nodes representing assessment results. The values of these assessment result nodes remain undefined (except the assessment result received for metric *M*) until a matching assessment result is received.

4.1.2 Aggregating the Evaluation Values

While different aggregation methods can be used for calculating the compliance values in the evaluation tree, the main method proposed in the initial proof of concept is setting the value of a node with a weighted arithmetic sum of the child nodes' values. The reason for choosing this approach is explained in Section 2.1. As shown in Figure 7, each tree node (representing an element in the standardisation hierarchy) has two configurable parameters: weight w and threshold T, and its value V is calculated using the weighted average of its child nodes:

$$V = \frac{\sum V_i w_i}{\sum w_i}$$

where i runs across the child nodes. Since the weighted sum is divided by the sum of weights, node values (and, consequently, thresholds) always fall in the interval $[0,1]$.

Thresholds simply mark the (un)conformity of a node by regarding nodes with $V \geq T$ as compliant. In the current proof-of-concept, thresholds are used mostly for visibility, to clearly display the nodes' (un)conformities to the user and to trigger the additional risk assessment evaluation of non-conformities. Another option would be to regard the nodes' values in their aggregation on the parent level as totally (un)compliant (0 or 1) depending on their compliancy with respect to the threshold. This way, the weighted aggregations would not propagate further than one level in the tree.

The evaluation tree can be easily simplified to an AND tree by setting all threshold values to 1.

The leaf nodes (representing assessment results) are expected to have logical Boolean values (evaluated by the Security Assessment components with respect to the evidence's compliance with the metric's target value), meaning that their values can only be 0, 1, or undefined (in case where no assessment results have been obtained for a specific metric-resource pair). Undefined values are regarded as uncompliant (0). As already mentioned, MEDINA defines metrics related to a particular requirement of the standard as a set of constraints which all need to be fulfilled to regard the requirement as compliant. For this reason, aggregation on the first level of the evaluation tree (from assessment results to compliance values of resources for a specific requirement) is done using the AND approach – resource nodes are assigned a value of 1 only if all metrics for a requirement are satisfied (or 0 otherwise).

Weights of individual elements can be assigned by the CSP (in collaboration with the auditor), possibly with inputs from the risk management framework according to the CSP's risk appetite.

If allowed by the specific standardisation scheme and chosen by the CSP (as well with inputs from the risk management), some elements of the scheme (nodes of evaluation tree) can be disregarded in the evaluation. In the example shown in Figure 7, one control of the standard is not selected and thus ignored in the aggregation to its parent node (control group).

Above we presented different methods that can be used in the Certification Evaluation Component in order to support various standards and certification schemes. The approach currently implemented in the MEDINA CCE component follows the aggregation method presented above. At this point, CCE does not receive any weights or thresholds of individual nodes and thus treats all parts of the certification tree equally in the aggregation. The distinction between the importance of various requirements or controls is considered by the *Risk*

Assessment and Optimisation Framework when determining how critical an incompliance is to the overall certification state of a CSP.

4.1.3 Operational Effectiveness

The state of the evaluation tree is saved after any assessment result is received that causes the tree nodes to change their value. Operational effectiveness measures were also added based on the statistics calculated on the tree states saved in a selected time interval. The CCE component exposes a gRPC function (queried by the *Life-Cycle Manager*) that calculates the following operational effectiveness measures for each node of the evaluation tree (see also Section 2.2):

- Cumulative durations a node was evaluated as compliant and as non-compliant
- The ratio of time the node was evaluated as compliant (vs. non-compliant)
- Minimal, maximal, and average Time-To-Fix (*Fail-Pass-Sequence-Duration*), meaning how long the CSP took to restore a control's compliance after its failure

This information is then processed by the *Life-Cycle Manager* (described in Section 6.2).

4.2 Implementation

4.2.1 Functional Description

This component collects assessment results and builds an evaluation tree representing the aggregated assessment results on higher levels of the certification scheme to determine compliance with the different certification elements.

Related requirements

All the related requirements (fully defined in D5.2 [18]) have been addressed and are Fully implemented in the second iteration of the CCE component.

Requirement id	CCCE.01
Short title	Continuous Evaluation of Assessment Results
Description	The evaluation component must be able to continuously evaluate incoming assessment results and integrate them into the overall certification evaluation.
Status	Fully implemented

Requirement id	CCCE.02
Short title	Evaluate the fulfilment degree per TOM
Description	The evaluation component must be able to evaluate continuously generated evidence and assessment results according to previously defined TOMs to calculate a degree of fulfilment.
Status	Fully implemented

Requirement id	CCCE.03
Short title	Configuration of needed metrics for requirements
Description	The evaluation component must be able to receive a selection of metrics needed to be satisfied for a particular requirement (as selected by the CSP) and consider it in the evaluation of requirements' fulfilment values.
Status	Fully implemented

Requirement id	CCCE.04
Short title	Fulfilment degree per control, group & entire certification
Description	The evaluation component must be able to aggregate the TOMs' fulfilment degrees to calculate the degree of fulfilment for controls, control groups, and the entire certification scheme.
Status	Fully implemented

Requirement id	CCCE.05
Short title	Temporal fulfilment degree per TOM
Description	The evaluation component should be able to evaluate continuously generated evidence according to previously defined TOMs to calculate a degree of fulfilment over time.
Status	Fully implemented

Requirement id	CCCE.06
Short title	Evaluate the time-to-fix indicator per TOM
Description	The evaluation component should be able to evaluate continuously generated evidence to calculate a time-to-fix indicator.
Status	Fully implemented

Requirement id	CCCE.07
Short title	APIs of the Continuous Certification Evaluation Component
Description	The evaluation component must provide APIs to the relevant WP3 components to provide measurement results, as well as to the *Risk Assessment and Optimisation Framework* and the *Life-Cycle Manager* to exchange relevant data.
Status	Fully implemented

4.2.1.1 Fitting Into Overall MEDINA Architecture

The data flow from gathering of technical and organizational evidence to the certificate life-cycle management is represented in Figure 5, showing the *Continuous Certification Evaluation* (CCE) in relation to the other components. Assessment results originating in the Security Assessment component(s) are forwarded to the CCE component through the *Orchestrator*. A single assessment result object contains an assessed value related to a specific metric (whether it is fulfilled or not) for a specific resource of the CSP's infrastructure.

The *Continuous Certification Evaluation* aggregates this information into an evaluation tree, which is stored (along its history) in the Certification evaluation storage database. The results are forwarded to the Risk Assessment component to further evaluate them and report possible deviations to the Automated Certificate *Life-Cycle Manager*. The Risk Assessment framework does not consume the entire tree, but only the bottom three levels of nodes (assessment results, resources, and requirements). As an additional metric in evaluating the final certificate state, the automated certificate *Life-Cycle Manager* can further inspect the operational effectiveness measures obtained directly from the CCE.

The *Continuous Certification Evaluation* component is also linked with the Catalogue of Controls and Metrics (developed in WP2) and the *Orchestrator* component. The Catalogue provides the structure of the used certification scheme (lists and mappings of metrics, requirements,

controls, control groups…), needed to construct the evaluation tree. The *Orchestrator* is the source of all configurations related to the evaluated service (Target of Evaluation), including the chosen controls/requirements and a list of monitored resources subject to evaluation.

4.2.2 Technical Description

The following subsections describe the technical details of the Continuous Certification Evaluation component.

4.2.2.1 Prototype Architecture

The CCE consists of the back-end (core) and front-end (web UI) components. The back-end CCE keeps and calculates evaluation trees and takes care of the connections with other MEDINA components. The web UI entirely runs on the client side (in the user's web browser) and, by interacting with the back-end API, displays all needed information to the user. The web UI is served by a simple Nginx server.

4.2.2.2 Components Description

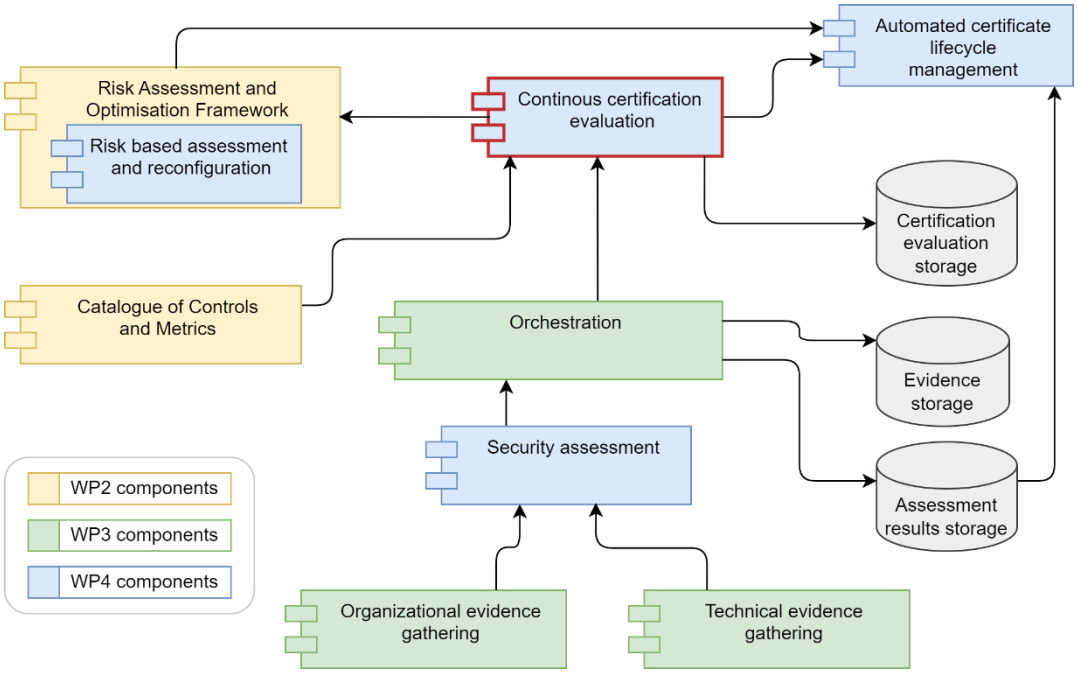

Figure 5. Continuous Certification Evaluation: diagram of interaction with related components

CCE back-end

The CCE exposes HTTP and gRPC APIs with distinct features. The gRPC API is used for receiving the assessment results from the *Orchestrator* and serving the historical evaluation statistics (operational effectiveness metrics) to the *Life-Cycle Manager*. The HTTP API is used to obtain the current and historic evaluation states by the CCE's web UI as well as the *Life-Cycle Manager*. The integrations are also implemented with the *Risk Assessment and Organisation Framework* (sending evaluation updates) and with the Catalogue of Controls and Metrics (receiving the certification framework schema).

The CCE includes a Certification Evaluation Storage database (implemented as a MongoDB instance) to store the evaluation state on every change. The history (as well as calculated operational effectiveness metrics) can be retrieved through the CCE's APIs.

CCE also includes support for multiple Targets of Evaluation (ToE). A ToE represents a cloud service being evaluated against a specified framework, set of controls, and another possible configuration. Thus, CCE holds current (and past) tree states for every ToE defined in the MEDINA framework.

CCE front-end

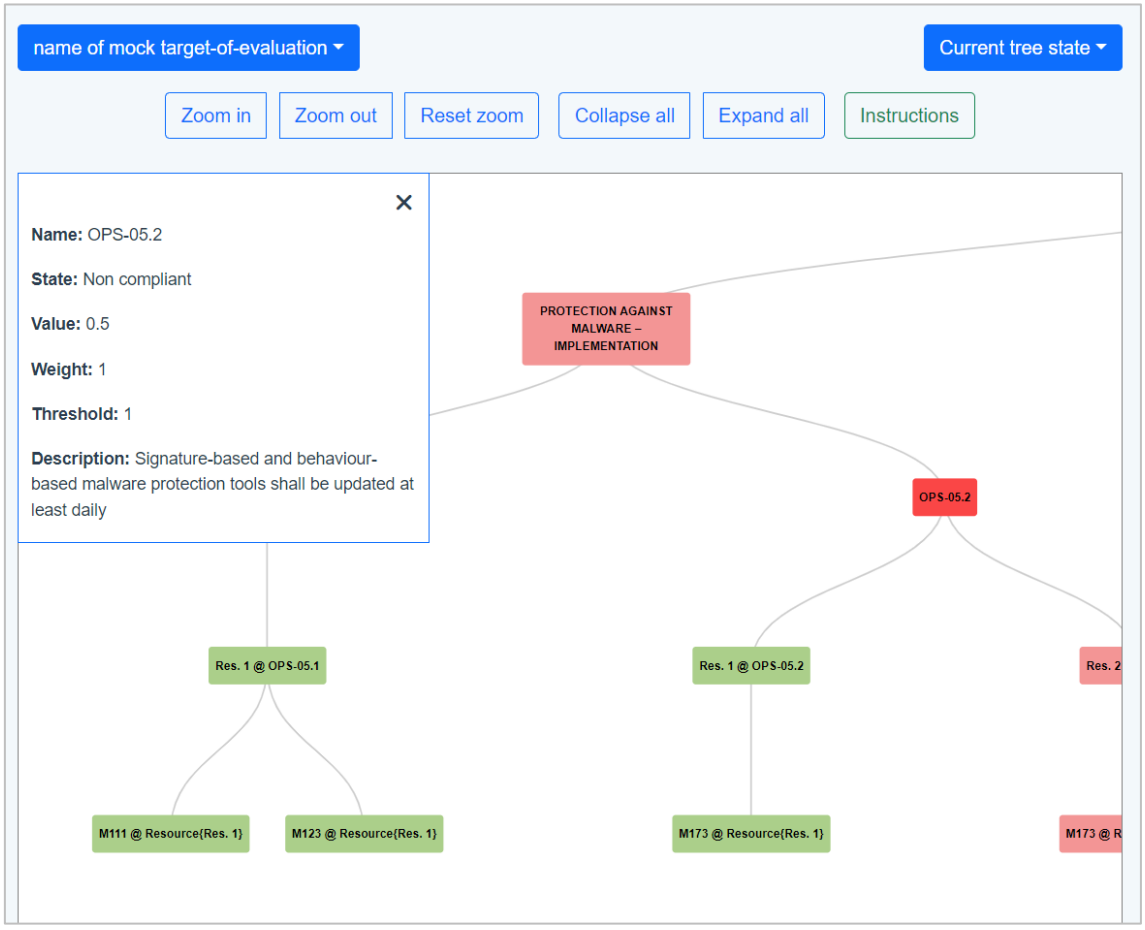

Figure 6. Screenshot of the CCE web UI

A web user interface for CCE is also implemented to enable a dynamic graphical overview of the current and past certification states. Figure 6 shows a screenshot of the interface. The evaluation tree is displayed graphically, and the user can expand parts of it to focus on a chosen set of controls. Nodes evaluated as compliant are shown in green, whereas non-compliant nodes are coloured in red. If a user has access to states for multiple Targets of Evaluation, they can switch between the views with the drop-down menu on the top left, and review the history for each of them with the top-right button.

4.2.2.3 Technical Specifications

The CCE back-end is built in Java using the Spring Boot framework, while the front-end is implemented in Javascript and the Vue.js framework. The back-end also uses a MongoDB database to store the history of evaluation tree states. The interfaces with the MEDINA Catalogue and the Orchestrator were implemented as standalone libraries and are thus also available for other component developers.

The CI/CD pipeline was established for both the core CCE and the web UI components to be automatically built and deployed in the MEDINA Kubernetes test environment.

4.3 Delivery and Usage

4.3.1 Package Information

The most important files and folders of the CCE (back-end) repository are presented in Table 2 below.

Table 2. Overview of the CCE back-end repository contents

File / folder	Description
kubernetes/	Contains Kubernetes definition files for automated deployment on the MEDINA Kubernetes dev & test clusters.
lib/	Contains jar libraries for interaction with the Catalogue and the Orchestrator.
src/main/proto/	Contains protocol buffer definition files (from which java classes for the gRPC API are built).
src/main/java/si/xlab/cce/	Contains the java source code.
Dockerfile	Contains code for building the component's Docker image.
docker-compose.yml	Contains code for easy deployment of all CCE components at once with Docker compose.
README.md	Contains details about installation requirements and instructions.
LICENSE	Contains a copy of the Apache 2.0 open-source license.

The most important files and folders of the CCE front-end repository are presented in Table 3.

Table 3. Overview of the CCE front-end repository contents

File / folder	Description
public/	Contains the index.html file.
src/	Contains all the javascript (Vue) source code.
package.json	Contains machine-readable instructions for installing javascript dependencies.
Dockerfile	Contains code for building the component's Docker image.
README.md	Contains details about installation requirements and instructions.
LICENSE	Contains a copy of the Apache 2.0 open-source license.

4.3.2 Installation Instructions

Both the CCE back-end and front-end services can be simply built and started as docker images. The configuration options for starting the containers are described in both repositories' README files. The CCE back-end repository also contains a docker compose file to ease installation.

4.3.3 User Manual

After starting CCE, the web server starts in the front-end docker container. After pointing the web browser to the address of this web server, the user is led to the web UI (see Figure 6) which can be simply navigated. The tree schema can be moved by dragging it with the mouse. The tree nodes can be clicked to display their details. Other functionalities are available through the visible buttons.

4.3.4 Licensing Information

The component is released open source with the Apache 2.0 license.

4.3.5 Download

The source code is available on the public MEDINA repository:

- CCE back-end (core): https://git.code.tecnalia.com/medina/public/continuous-certification-evaluation
- CCE front-end (web UI): https://git.code.tecnalia.com/medina/public/cce-frontend
- Java library for communication with the Catalogue of Controls and Metrics: https://git.code.tecnalia.com/medina/public/catalogue-client-java
- Java library for communication with the MEDINA Orchestrator: https://git.code.tecnalia.com/medina/public/orchestrator-client-java

4.3.6 Summary and Future Work

4.3.6.1 Summary

In this section, we presented the *Continuous Certification Evaluation* (CCE). This component evaluates the fulfilment degrees on various levels of the certification tree, representing the certification requirements, controls, control groups, etc. The component was entirely built in the scope of MEDINA project.

4.3.6.2 Limitations and Future Work

The evaluation tree built by the CCE component is an enhanced representation of data coming from the evidence gathering and security assessment tools. The confidence of the CCE's outputs thus largely depends on the data provided by those components.

The CCE can be efficiently used to review the state of gathered evidence at the chosen point in time, but a limitation is that no conclusions about the actual risk state or the certification status can be made solely based on the CCE outputs. Other components of the MEDINA solution (*Risk Assessment and Optimisation Framework* and certificate *Life-Cycle Manager*) help users to better understand the broader view of their certification state.

The future work planned for the CCE implementation includes:

- integration with the MEDINA Single-Sign-On system and support for user authentication and authorization,
- changes according to updates in the common data model (to enable full and dynamic support for multiple targets of evaluation),
- finalizing integration with the *Orchestrator*, mostly regarding the exchange of data about targets of evaluation and their configuration (e.g., requirements to be covered).

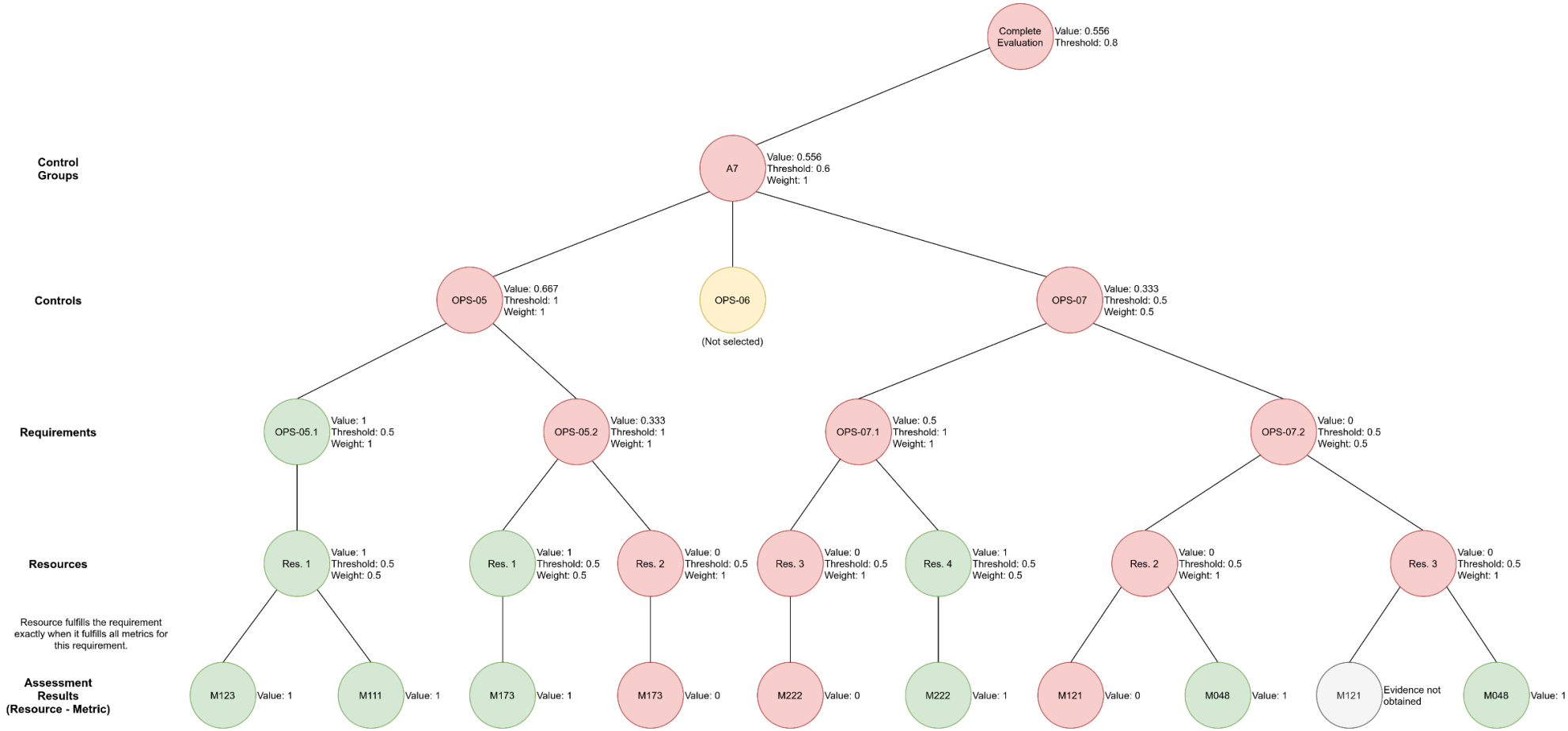

Figure 7. An excerpt of an example evaluation tree representing (non-)conformities of standardisation hierarchy elements

5 MEDINA Establishment of a Digital Audit Trail

This section sets the theoretical and methodological background for establishing a digital audit trail in MEDINA with the goal of increasing the trustworthiness of the overall framework through the *MEDINA Evidence Trustworthiness Management System developed in WP3*. Section 5.1 first presents a risk assessment with regard to storing evidence and assessment results. Section 5.2 then discusses different measures to address these risks, focusing on Blockchain technologies. Section 5.3 then presents different hashing techniques since they are essential in storing information about evidence and assessment results. Finally, Section 5.4 describes different alternatives to verify the integrity of evidence and assessment results.

5.1 Risk Assessment

The purpose of this risk assessment is to identify threats and vulnerabilities, and to identify different ways to mitigate the resulting risks. The risks are assessed following a standard methodology (see e.g., Torr [19], or the NIST guidelines [20]). First, we state assumptions regarding the MEDINA framework. Then we identify the assets to be protected, their protection goals, as well as the users of the system. We continue by defining the attacker model and then model possible attack vectors. Risks are then assessed based on the likelihood of an attack and their potential impact. Finally, we discuss the possible mitigations.

5.1.1 Assumptions

It is important to highlight some assumptions to be considered for this risk analysis. In this case, all MEDINA tools are considered reliable**; we trust all MEDINA tools**. As a result, MEDINA tools cannot be considered as threats, nor any vulnerability could be detected in their implementation and/or operation. Furthermore, human factors are not considered in this analysis, e.g., social engineering-based attacks.

5.1.2 Asset Classification Scheme

The first step in a risk assessment process is to identify and define all valuable assets in scope. This risk analysis is focused on **critical data**, or other data whose exposure would have a major impact on the MEDINA framework operation.

Table 4. Overview of types of data and their sensitivity levels

Type of data	Description	Level of sensitivity
Evidence (for more details, see trustworthy evidence data model [3])	• id • toolId • resourceId • cspid • measurementResult • timestamp	• The field *measurementResult* has a high level of sensitivity. • The rest of the fields has a low level of sensitivity.
Assessment Results	• id • metricId • assessmentResult • complianceResult • associatedEvidencesId • timestamp	• The fields *assessmentResult* and *complianceResult* have a high level of sensitivity. • The rest of the fields have a low level of sensitivity.

5.1.3 Potential Users

Next, it is recommended to identify and describe who is going to operate or access the assets identified in Section 5.1.2 as critical data.

Table 5. Overview of the different users in MEDINA

User	Data	Access Level	Number of users	Organization
Orchestrator	• Evidence • Assessment Results	Full (Read and Write)	One instance per organization	Internal organization
Organization employees	• Evidence • Assessment Results	Read only	Undetermined	Internal Organization
Auditors	• Evidence • Assessment Results	Read only	Undetermined	CAB

5.1.4 Protection Goals

Information assurance is an approach of managing risks related to the use, processing, storage, and transmission of information or data. The three main protection goals are confidentiality, integrity, and availability (CIA). Additionally, authenticity, authorization and non-repudiation can be considered.

Confidentiality: Confidentiality is the property that guarantees information is not made available or disclosed to unauthorized individuals. Assets gathered in Section 5.1.2 represent sensitive information that should be kept private from all unauthorised users; confidential information must only be accessed by authorized users. **In MEDINA, confidentiality is a must**, since evidence and assessment results contain sensitive information about the security posture of the audited service provider.

Integrity: Integrity is the property of safeguarding the accuracy and consistency of assets; it means that information cannot be altered or tampered with, ensuring the data correctness, and protecting against unauthorized modification. In MEDINA, auditors need to trust the stored data regarding its integrity to provide corresponding certificates. For that reason, **in MEDINA, integrity is a must**.

Availability: Availability is the property of being accessible and usable upon demand. Availability assumes that information systems, as well as the information itself, is available and operating as expected when needed or requested. In MEDINA, evidence and assessment results should be available for the proper *Orchestrator* operation as well as for auditors to verify them when needed. Consequently, **although it is not a must in MEDINA, it is highly recommended to guarantee evidence and assessment results availability**.

Authenticity: Authenticity is the property that guarantees an entity is what it claims to be, proving that all parties involved in an action are who they claim to be. It is of great importance to ensure the genuineness of every asset, reducing instances of fraud by way of misrepresentation.

MEDINA needs to authenticate all the information sources to certify who provided, modified, or even deleted certain data related to evidence and/or assessment results. This way, auditors can be sure that trusted sources have operated the MEDINA framework and no impostor source has ever replaced legitimate sources. **In MEDINA, authenticity is a must.** However, due to the assumption based on trusting all the MEDINA tools, authenticity of evidence and assessment results is given. Anyway, some additional secure authentication mechanisms, such as mutual authentication between different MEDINA components, could be added so that anyone outside the MEDINA system could provide information.

Authorization: Authorization is the property that determines access levels or user privileges related to system resources including information. It is related to the access control techniques, granting, or denying access to a specific resource depending on the user identity. **This is not the**

role of MEDINA as MEDINA is a framework used by auditors, with the same access levels or user privileges. All potential MEDINA users will have the same role: auditors.

Non-repudiation: Non-repudiation is the ability to prove an event or action has occurred as well as to identify its originating entities in order to resolve disputes about the occurrence or non-occurrence of the event and who were the involved entities. In MEDINA, non-repudiation is very relevant in two senses. On the one hand, sources providing data (evidence and/or assessment results) should not be able to deny their involvement in MEDINA; once they provide data, they cannot deny their data provision. However, due to the assumption based on trusting all the MEDINA tools, non-repudiation is already guaranteed. On the other hand, sources accessing data (evidence and/or assessment results) should neither be able to deny their involvement in MEDINA; once they access data, they cannot deny the have read the data.

5.1.5 Potential Attackers

It is important to develop a catalogue of potential attackers, in other words, threat sources. There are two main types of attackers: outsiders and insiders.

In general, outsiders can be classified based on their professional level: organized attackers, hackers, and amateurs.

- Organized attackers (terrorists, nation states, and criminals). They are generally highly trained, highly funded, and are often backed by substantial scientific capabilities. In many cases, their highly sophisticated attacks are directed toward specific goals.
- Hackers: they may be perceived as benign explorers, malicious intruders, or computer trespassers. In most cases, they are highly trained and could be sponsored by criminal organization or governments for financial gain or political purpose.
- Amateurs: these are less-skilled hackers, also known as "script kiddies" who often use existing tools that can be found on the Internet. They are not as dangerous as the previous ones since they do not have the ability to create their own, adapted tools.

In general, insiders are people from the own organization (or with a strong relation with the organization) who have skills, knowledge, resources, and access to the organization systems. Consequently, malicious insiders will have a deep knowledge of the MEDINA framework.

It is recommended to identify threats to the MEDINA framework regarding the identified protection goals and attacker types, from security breaches to human errors.

Table 6. Overview of main potential threats from different attackers

Attacker	Threat Action
Outsiders	• System intrusions • Identity theft
Malicious insider	• Browsing of personally identifiable information. • Unauthorized system access through escalation of privilege. • Accidental or ill-advised data modification/deletion • Accidental or ill-advised actions taken by employees that result in unintended physical damage, system disruption, etc.
Environmental	• Natural or man-made disasters; HW failure, etc.

Also, it is recommended to identify potential attackers' motivations to determine the real risks.

Table 7. Overview of main motivations for different attackers

Attacker	Motivation
Outsiders	• Someone who wants to change data to ensure the certificate is not obtained by the organization. • Someone who wants to obtain sensitive information (e.g., for espionage). • Unhappy customers who want to damage the organization (discredit, loss of customers, etc.). • Intellectual challenge. • Social/political/economic incentive.
Malicious insider	• Someone who wants to change data to successfully obtain a certificate. • Unhappy workers who want to damage the organization (discredit, loss of customers, etc.). • Someone who makes a mistake modifying or deleting information (trusted employees accidentally misplacing information).
Environmental	• N/A

5.1.6 Potential Attacks

It is essential to assess which vulnerabilities and weaknesses could allow potential attacks breaching the MEDINA framework security.

Table 8. Description of the main potential attacks in MEDINA

Protection goal	Potential attack	Description
Confidentiality	• Eavesdrop on database connection • Eavesdrop on tool connection	Secretly listen to the private communication between the gathering/assessment tools and the *Orchestrator* and between the *Orchestrator* and the database without consent to gather data (or metadata) information. It is usually related to a lack of encryption services.
	Gain read access to database	Broken access control vulnerabilities exist when a user can access specific data that they are not supposed to be able to access. It is related to not enforcing any protection over sensitive data or by means of privilege scalation.
	Phishing	Obtain authentication data by impersonating oneself as a trustworthy entity in order to gain access to private data.
Integrity	• MitM attack on database connection • MitM attack on tool connection	The attacker secretly relays and alters the information in the communication between the gathering/assessment tools and the *Orchestrator* and between the *Orchestrator* and the database who believe that they are directly communicating with each other.
	Gain write access to database	Broken access control vulnerabilities exist when a user can access specific data that they are not supposed to be able to access. It is related to not enforcing any protection over sensitive data or by means of privilege escalation.

Protection goal	Potential attack	Description
Availability	DoS attack to the database	Flooding the database with traffic or sending it information that triggers a crash in order to shut down the system, making it inaccessible to its users. There is a special risk with centralized systems (Single point of failure).
	Internet access down	Internet outage due to an external problem (natural disaster, etc.)
	Gain write access to database	With write access to the database, an attacker can simply delete evidence and assessment results (see the integrity threat).
Authenticity	Phishing for private key (credentials) for database access theft	Obtain authentication data by impersonating oneself as a trustworthy entity in order to gain access to private data.
	Poor private key (credentials) strength for database access	Passwords used are weak. Attackers could guess the password of a user to gain access to the database.
	• MitM attack on database connection • MitM attack on tool connection	The attacker secretly relays and alters the information in the communication between the gathering/assessment tools and the *Orchestrator* and between the *Orchestrator* and the database who believe that they are directly communicating with each other.
Non-repudiation	Phishing for private key (credentials) for database access theft	Obtain authentication data by impersonating oneself as a trustworthy entity in order to gain access to private data.
	Poor private key (credentials) strength for database access	Passwords used are weak. Attackers could guess the password of a user to gain access to the database.
	• MitM attack on database connection • MitM attack on tool connection	The attacker secretly relays and alters the information in the communication between the gathering/assessment tools and the *Orchestrator* and between the *Orchestrator* and the database who believe that they are directly communicating with each other.

5.1.7 Likelihood of Exploitation

The next step involves determining the likelihood of the potential attacks identified in Section 5.1.6 resulting in succeeding against our system. Likelihood is the probability that a vulnerability is exercised in an attack. It mainly depends on the attackers' motivation and capacity, the nature of the vulnerability, and the existence of countermeasures.

Probability can be ranked as:

- **High**: The attacker is highly motivated and sufficiently capable; controls to prevent the vulnerability to being exercised are inefficient.

- **Medium**: The attacker is motivated and capable, but controls are in place that may impede successful exercise of the vulnerability.
- **Low**: The attacker lacks motivation and/or capability, or controls are in place to prevent or, at least, significantly impede the vulnerability for being exercised.

Table 9. Likelihood of different attacks to happen

Potential attack	Likelihood
Eavesdrop on database connection	Medium
Eavesdrop on tool connection	Medium
Gain read access to database	High
Phishing	Medium
MitM attack on database connection	Medium
MitM attack on tool connection	Medium
Gain write access to database	High
DoS attack to the database	High
Internet access down	Low
Phishing for private key (credentials) for database access theft	Medium
Poor private key (credentials) strength for database access	High

5.1.8 Impact

The next step in a risk analysis is to perform a risk impact analysis to understand the consequences of an incident. The impact will be used to calculate and prioritize risks in the final step.

- **High impact**: There is a strong need for corrective measures.
- **Moderate impact**: Corrective actions are needed, and a plan must be developed to incorporate these actions within a reasonable period of time.
- **Low impact**: It must be determined whether corrective actions are still required or decide to accept the risk.

Table 10. Overview of effect and impact of the potential attacks

Effect	Impact
Evidence/Assessment results will not be trustworthy	High
Evidence/Assessment results will not be available	Moderate
Audit will not be trustworthy	High
Organization discredit	High
Unfair competence	High

5.1.9 Risk Calculation

The last step in a risk assessment is to combine the likelihood and the impact values to derive a risk value. Table 11 shows the risk value for the potential attacks identified in Section 5.1.6.

Table 11. Overview of the risk of the potential attacks

Potential attack	Likelihood	Impact	Risk
Eavesdrop on database connection	Medium	Moderate	Moderate
Eavesdrop on tool connection	Medium	Moderate	Moderate
Gain read access to database	High	Moderate	Moderate
Phishing	Medium	Moderate	Moderate
MitM attack on database connection	Medium	High	High

Potential attack	Likelihood	Impact	Risk
MitM attack on tool connection	Medium	High	High
Gain write access to database	High	High	Very High
DoS attack to the database	High	High	Very High
Internet access down	Low	High	Moderate
Phishing for private key (credentials) for database access theft	Medium	High	High
Poor private key (credentials) strength for database access	High	High	Very High

5.1.10 Security Requirements

Based on the risk analysis, mitigation techniques are needed to reduce or even mitigate the identified risks.

- A set of rules are required to be applied to limit access to personal data only to authorized people → User access control: identification & authorization.
- Therefore, data should be kept secure applying Privacy Enhancing Technologies (e.g., encryption, pseudonymization, anonymization, identity, and access management).
- A set of rules are required to ensure the data is trustworthy and accurate.
- A set of rules are required to prevent accidental disclosure of sensitive data.

Taking these security requirements into consideration, **a secure *MEDINA Evidence Trustworthiness Management System* for audit trail will be included in the MEDINA framework.**

5.2 Solutions for Audit Trails

As presented in Section 2.4, Blockchain technology has started to be considered as a suitable technology for auditing purposes due to some of its main features: decentralization, trustlessness, transparency, traceability, immutability, and security. However, other options, such as traditional databases or replicated databases could be also considered. *Appendix A: Alternatives to Blockchain for Audit Trails* includes a comparative analysis of Blockchain with traditional and replicated databases, **concluding that Blockchain is suitable for audit trails**. Moreover, *Appendix B: Blockchain Technologies* compares different Blockchain solutions concluding that **a private Blockchain network is more suitable for the audit trails**.

Taking these two ideas into consideration, and the analysis of Blockchain-related technologies from *Appendix B: Blockchain Technologies*, the technologies whose features better fit *MEDINA Evidence Trustworthiness Management System* requirements are: **Hyperledger Fabric** [21] and **Quorum** [22] (traditional general purpose private Blockchains).

Hyperledger Fabric aims to provide the basis for an extensible, modular, business-focused architecture that can be adopted by organizations in a variety of sectors. In contrast, Quorum is presented as an application-independent platform, with numerous differences and adaptations with respect to Ethereum but focusing on business needs. Therefore, although different in their initial approaches, both technologies aim to solve the problems associated with consortiums of professionals and organizations.

Table 12 presents a comparison between Hyperledger Fabric [21] and Quorum [22], the most known private technologies.

Table 12. Overview of the most suitable Blockchain technologies features for MEDINA audit trail

Feature	Hyperledger Fabric	Quorum
Description	Modular Blockchain platform	Distributed registration protocol for enterprises and Smart Contracts platform.
Governance	Linux Foundation	J.P. Morgan (now, ConsenSys)
Operation mode	Permissioned (private)	Permissioned (private)
Participation	Per organization	Per node
Permission level	Fine grained (creation of users, deployment of Smart Contracts...)	Simple (validating node or not)
Message privacy	Yes	Yes
Type of privacy	By communications channel	By transaction
Private communications	Establishment at the beginning. Difficult to dissolve	Indicated in each message. No fixed link
Consensus	- SOLO (ordering) - Kafka (ordering) - Simplified BFT (future) - Practical BFT (future)	- Raft (no BFT) - Istambul BFT
License	GPL / LGPL	GPL / LGPL
Confirmation time	Instant	Instant
TPS	450-900 (theoretical)	800 (theoretical)
Transaction logs	Hash-linked blocks	Hash-linked blocks

As it can be deduced from the previous table, both technologies have similar features that can be useful. For that reason, and only considering the simplicity in network management, **Quorum has been considered as the Blockchain network technology for the *MEDINA Evidence Trustworthiness Management System***.

5.3 Guarantee of Data Integrity: Hash Functions

In MEDINA, hashes are used within the *MEDINA Evidence Trustworthiness Management System* in two ways:

- They are a fundamental part of the Blockchain technology considered as backbone.
- They have been considered a secure way for guaranteeing information integrity without breaking privacy required by sensitive information (such as evidence and assessment results).

5.3.1 What is a Hash?

A cryptographic hash function is a mathematical algorithm that transforms any incoming data into a series of output characters. A hash is the result of a hash function whose primary purpose is to encode data to form a unique string of characters regardless of the amount of data initially entered into the function [23]. In other words, any input data always generates the same output hash while any minimum change on the input results in a totally different output hash, as it is shown in Figure 8.

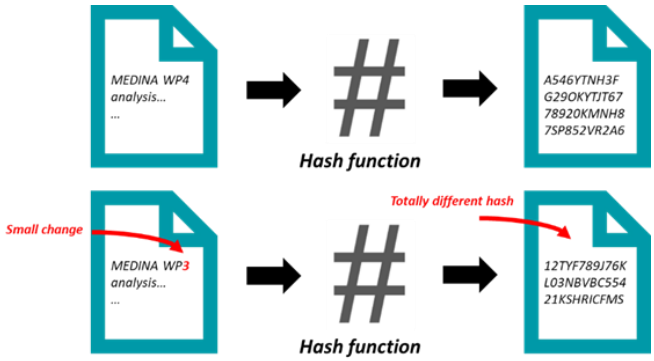

Figure 8. Hash functionality (source: MEDINA's own contribution)

The first data input results in a unique hash. In the second entry, a small modification has been made to the text. This, although minimal, completely alters the result of the hash. This shows that the hashes will be unique in any case, which allows to be sure that no malicious actor can easily crack the hashes. Although this is not impossible to achieve, a hacker could spend hundreds of years processing data to do so. It is these two observations that provide the security for using this method in different sensitive areas. Digital certificates, unique signatures of sensitive or secret documents, digital identification and key storage are some typical use cases.

For a simpler understanding, a simple and everyday example can be considered: the making of a cake. Each of the cake's ingredients would be the equivalent of the data input. The process of preparing and baking the cake would be the process of encoding the data (ingredients) by the function. At the end, we obtain a cake with unique and unrepeatable characteristics given by the ingredients of the cake. While the opposite process (taking the cake to its initial ingredients state), is practically impossible to perform.

The first hash function dates back to 1961. At that time, Wesley Peterson created the Cyclic Redundancy Check function [24]. It was created to check the correctness of data transmitted over networks (such as the Internet) and digital storage systems. Easy to implement and very fast, it gained acceptance and is today an industry standard. With the evolution of informatics and computers, these systems became more and more specialized.

5.3.2 Properties of a Good Hash Algorithm

Desirable features for a good hashing algorithm are summarized in the following [25] [26]:

- **Easy to compute:** Hash algorithms are very efficient and do not require much computing power to run.
- **Determinism:** A hash algorithm must be deterministic, in other words, it must always give an output of identical size, regardless of the size of the input data. This means that, if a single sentence is encoded, the resulting output hash must be the same size as when encoding an entire book.
- **Irreversibility:** A strong hash algorithm is one that is pre-image resistant, meaning that it is not feasible to invert a hash value to recover the original input data ("one-way functions"). The only way to obtain the input data is by brute-forcing all possible inputs.
- **Collision resistance:** A collision happens if two unique samples of input data result in identical output hashes. In other words, the concern is that someone could create a malicious file with an artificial hash value that matches a genuine (secure) file and pass it off as real because the output hash would match. Therefore, a good and reliable hash algorithm is one that is resistant to these collisions.

- **Avalanche effect:** Any change made to the input data, no matter how small, will result in a massive change in the output hash. Essentially, a small change (such as adding a comma) becomes something much larger, hence the term "avalanche effect".
- **Hash speed:** Hash algorithms should run at a reasonable speed. This property, however, is a bit more subjective. Faster is not always better because the speed should depend on how the hash algorithm will be used: sometimes, a faster hash algorithm is desired (for website connections, for example), and other times it is better to use a slower one that takes longer to execute, e.g., for password hashing to prevent brute-forcing.

Please note that we describe several existing hash functions in *Appendix C: Current Leading Hash Algorithms*, concluding that **SHA-2 is considered the hash function with best trade-off between security and performance**.

5.3.3 Are hashes completely irreversible?

Hash functions aim to be irreversible and therefore the result of applying a hash function to specific data should avoid re-identification. Despite this, the determinism feature that is implicit in hashing processes increases the probability of identifying the original data from the hash.

For example, let´s consider a phone number of 9 digits (9 bytes, 72 bits) of which a 64-bit hash is calculated [27]. The phone number space is larger than the hash space. If all possible hashes were calculated for all 72-bit combinations, inevitably the entire hash space would be covered, and several collisions would occur as the hash function would have to "compress" the 72 bits of the number into the 64 bits of the hash. However, with a deeper analysis, the 9 digits are numerical, which means 1000 millions of combinations. It seems a very large number, but translating it into bits, the amount of data has been reduced from 72 bits to approximately 30 bits. That is, much less than the initial 72 and already below 64 bits of the size of the hash.

If dealing with Spanish cell phone numbers, for example, they will start with either 6 or 7. Therefore, since the first number is fixed, there are only 200 million combinations (approximately 28 bits). But there are not 200 million subscribers in Spain. The number of current mobile lines is actually less than 60 million (26 bits). In fact, the operator with the highest number of mobile lines is less than 20 million (20 bits of information). Consequently, the number of combinations decreases enormously, and the real information contained in the original 72 bits data is only around 20 bits of information.

Given that a desktop computer can calculate more than 1 million hashes per second, a dictionary can be created for all possible hashes of a given operator's phones in less than 20 seconds, practically in real time. In other words, the information referenced by the hash can be reversed and security will be broken. In this example, the amount of information was small, but even for much larger message spaces, with more information, it is possible to retrieve the information referenced by the hash within an acceptable time even for much larger message spaces, with more information thanks to techniques known as Rainbow Tables [28] that allow the reversal of the hashes (reversibility).

When input data has an implicit order, less real information it contains and the set of possible messages (message space) is greatly reduced, which facilitates message reversibility (re-identification). Therefore, it is necessary to distinguish between the data in a message (72 bits in the example) and the information contained in the data (20 bits in the example).

The degree of order (or disorder), of a data is known as entropy. The higher the entropy, the more information a data set will contain. The smaller the message space, and the lower the entropy, the lower the risk of collision in hashing, but re-identification will be more likely.

Conversely, the higher the entropy, the greater the chance of a collision, but the risk of re-identification will be much lower.

Therefore, the measure of the amount of information, which is quite different from the number of bits that is being used to record a message, is one of the most important analyses that needs to be performed whenever a message is to be protected.

5.3.4 Hashes in MEDINA

5.3.4.1 Blockchain

Hash functions are widely used within Blockchain technology because they are fast, efficient, computationally inexpensive, and unique. When Satoshi Nakamoto published his Bitcoin whitepaper [29], he explained why and how to use SHA-256 and RIPEMD-160 in Bitcoin. Since then, Blockchain technology has evolved a lot, but the basics remain the same: make use of strong cryptography and hashes to make the technology secure and private.

The most important uses of hash functions in Blockchain are:

Address creation (Address Wallet): The addresses of cryptocurrency wallets are a secure representation of the wallet's public keys. Public keys are usually very long and complex. It is for this reason that Blockchains usually use hash functions to derive a shorter address. This process is used at various times to shorten the address and add an extra layer of security. For example, in Bitcoin, the process of creating a wallet address, uses the hash functions RIPEMD-160 and SHA-256.

Mining Process: The mining process is another important stage of blockchain technology where hash functions are used. For example, in Bitcoin, mining makes intensive use of SHA-256 hashes calculation in a distributed way in each of its nodes. Miners are responsible for calculating millions of hashes to create new Bitcoin blocks. The process is also used to verify transactions made on the network. While the process of calculating hashes is very fast, its intensive use hinders the process drastically. This leads miners to use high computational power to solve Bitcoin puzzles.

Smart Contracts: This is another area where hash functions are heavily used. Blockchains such as Bitcoin, Ethereum, NEO or TRON make use of Smart Contracts to power different applications. These applications are driven by a public contract between parties. A public contract has a unique hash that is given by what the contract says. If the contract is modified, the old contract is terminated and a new one is generated with a new hash. In this way the hash determines the correct contract to use within the decentralized application, facilitating its control.

Another use of hashes in smart contracts is to guarantee the validity and authenticity of the contract. For example, a contract for the sale of a house with a payment made in cryptocurrencies. The definition of the contract and its hash are unalterable witnesses of sale made between two parties.

As it can be deduced, each Blockchain technology uses specific hash functions by design.

5.3.4.2 Evidence (and Assessment Result) Integrity

The *MEDINA Evidence Trustworthiness Management System* is used for providing a guarantee of integrity of evidence and assessment results. However, this information is considered sensitive and should not be stored in a “public” storage such as Blockchain which does not allow future deletions. For this reason, information is not directly stored in Blockchain, instead, hashes for the different evidence and assessment results values are stored on the Blockchain, as they

do not disclose any input data but can be useful for guaranteeing that information has not been altered. From section 5.1.2, the specific features to be protected with hashes as they are considered sensitive are *measurementResult* from the evidence and *assessmentResult* and *complianceResult* from the assessment result.

The idea is that when evidence or assessment results are obtained, the *Orchestrator*, i.e., the MEDINA component in charge of storing evidence and assessment results on the Blockchain, automatically calculates the associated hash. By this way, the hash of the original information is recorded on the Blockchain and cannot be altered by design. **SHA2-256 has been considered suitable for MEDINA as it is widely standardized, it is secure enough and with a good trade-off between security and performance.**

Later on, when an auditor needs to verify if an evidence or assessment result has been modified, the current data value can be retrieved, the hash function can be applied, and the result can be compared with the hash value previously stored in the Blockchain, to ensure that the data has not been modified.

As it has been stated in Section 5.3.3, input data should have a high entropy in order to avoid revealing the input data. The entropy of the input data for hashing in MEDINA is:

- *measurementResult:* This refers to the specific evidence gathered for fulfilling a specific EUCS security requirement. There is no specific size, so its entropy is considered high by default.
- *assessmentResult:* This refers to the specific assessment result for a specific EUCS security requirement. There is no specific size, so its entropy is considered high by default.
- *complianceResult:* This refers to the compliance result. In this case there are only two possible values: compliance or not compliance. For this reason, the entropy is extremely low, and security could be easily broken. For this reason, entropy must be increased by adding a **random 256 bits value** to the *complianceResult*. By this way, entropy is increased, and security is enhanced.

5.4 Verifying Evidence and Assessment Results

The starting point is that an audit is taking place (both internal or external audit). Some results (evidence and assessment results) have been obtained from the audit process and the objective for the auditors is to verify them with the evidence and assessment results hashes values recorded on the Blockchain so as to guarantee that they have not been tampered with.

Evidence and assessment results are not directly stored in the Blockchain, as they are considered sensitive information for the CSPs. Instead, evidence and assessment results hashes are stored, avoiding data disclosure but ensuring data integrity as:

- Each evidence/assessment result value is assigned a specific hash. Any slightest change in the evidence/assessment result value will result in a change in the hash.
- If the hash does not change, the evidence/assessment value has not been tampered; if the hash changes, the evidence/assessment value has been tampered (we do not know the specific change in the value, but we know that it has changed. By this way, current evidence/assessments results cannot be considered as valid.

There are several ways auditors could make this verification through the MEDINA framework, depending on the point the current evidence/assessment results hashes are obtained to be compared with those recorded on the Blockchain. The next subsections analyse different options.

5.4.1 Calculation of Hashes in the Orchestrator

The auditors directly obtain the evidence/assessment results hashes from the *Orchestrator* (using its graphical interface) and compare them with the hashes recorded on the Blockchain. There are different options:

- The *Orchestrator*, which has calculated the current evidence/assessment results hashes, can directly verify these values with those recorded on the Blockchain through the *MEDINA Evidence Trustworthiness Management System* API, as it is shown in Figure 9. As a result, a TRUE or FALSE indication would be received on the *Orchestrator* and would be shown through the *Orchestrator* user interface.

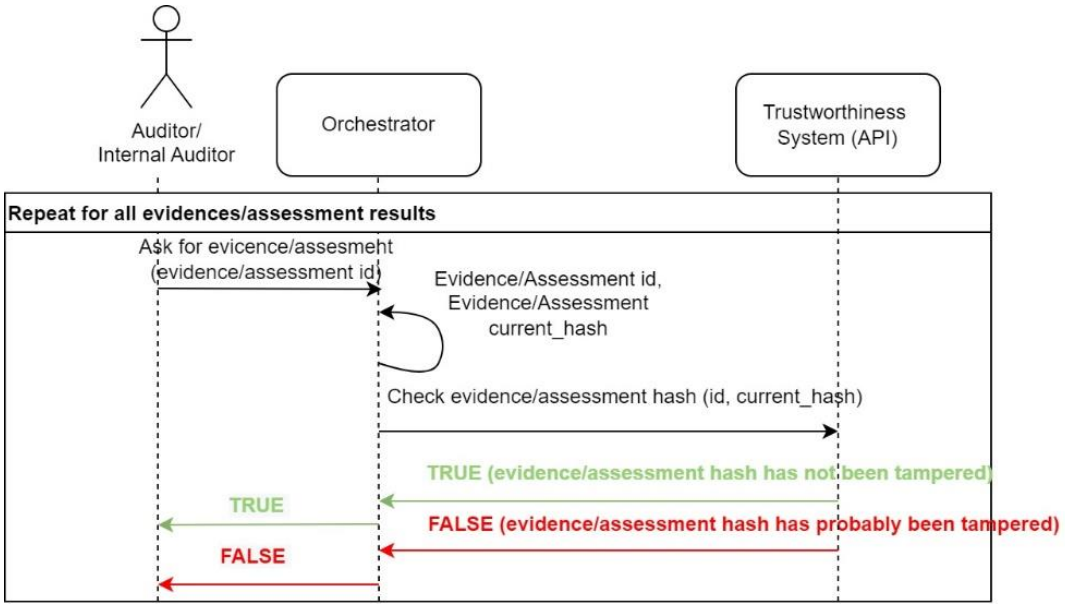

Figure 9. Automatic verification from the Orchestrator using the MEDINA Evidence Trustworthiness Management System API

The main advantage of this solution is that the complete process is automatically done for the auditors: they only need to provide evidence/assessment result and the *Orchestrator* would automatically complete the whole process for them, resulting in a TRUE/FALSE result.

The main disadvantage is that the *Orchestrator* would need to be modified to support the evidence/assessment result id indication and the verification result obtaining through its user interface. Additionally, the *Orchestrator* internal implementation would also need to include the hash obtaining and the checking evidence/assessment hash call to the *MEDINA Evidence Trustworthiness Management System*. However, the *MEDINA Evidence Trustworthiness Management System* would not need any update.

- The *Orchestrator* provides the current evidence/assessment results hashes to the auditors who, manually, look for the specific hash on the user interface of the *MEDINA Evidence Trustworthiness Management System* as shown in Figure 10. If it is found, the *current_hash* is correct, as it is very difficult to obtain the same hash from two different sets of data. A second option is that the auditors manually look for the hash recorded on the Blockchain associated to the specific evidence/assessment result id through the user interface of the *MEDINA Evidence Trustworthiness Management System* and, manually, compare it with the *current_hash* received from the *Orchestrator*.

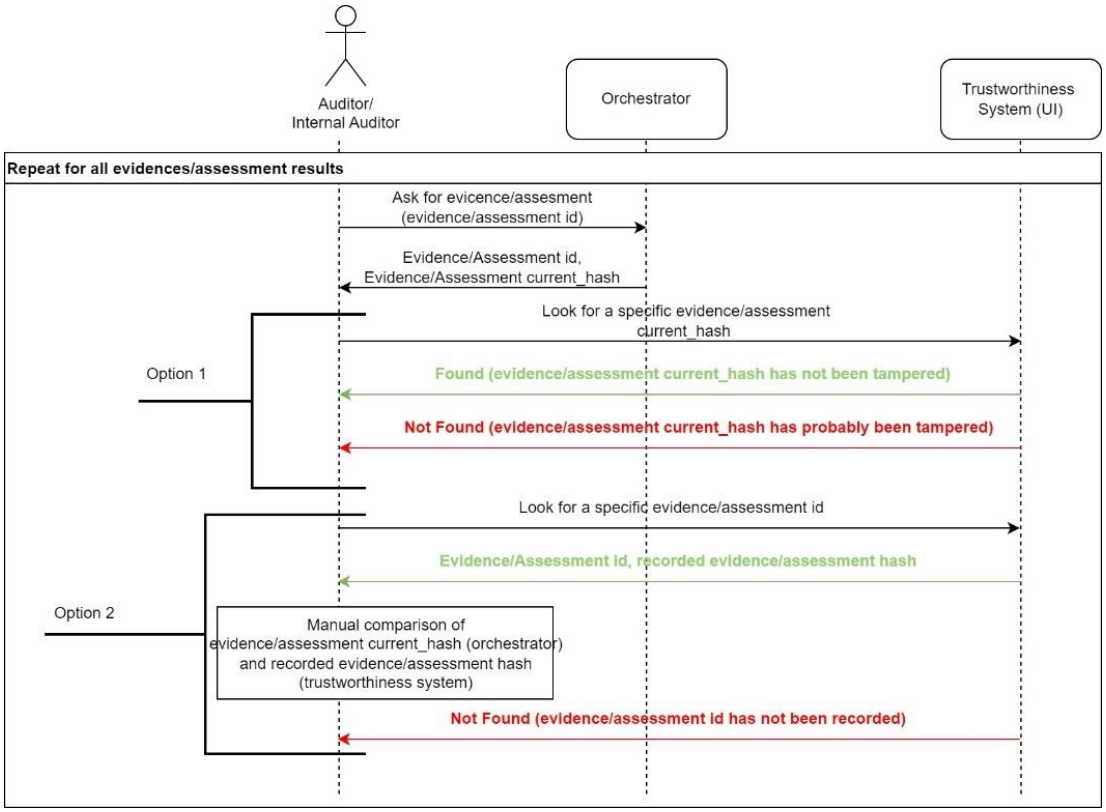

Figure 10. Manual verification from the Orchestrator using the MEDINA Evidence Trustworthiness Management System GUI

The main advantage of this solution is that the auditor itself makes the verification and does not need to trust on the *Orchestrator* (security increases).

The main disadvantage is that the *Orchestrator* would need to be modified to support the evidence/assessment result id indication and the hash obtaining. The *MEDINA Evidence Trustworthiness Management System* would not need any update. Another disadvantage is that this is a manual process for the auditors.

5.4.2 Calculation of Hashes in the MEDINA Evidence Trustworthiness Management System

The auditors obtain the evidence/assessment values from the *Orchestrator* and use the *MEDINA Evidence Trustworthiness Management System* for the hashes collection and the verification in comparison with hashes recorded on the Blockchain. There are different options:

- Once the current evidence/assessment results values are obtained from the *Orchestrator*, the auditors will use the *MEDINA Evidence Trustworthiness Management System* user interface for obtaining the associated *current_hash* values. They will then manually look for this specific value on the *MEDINA Evidence Trustworthiness Management System* user interface as shown in Figure 11. If the *current_hash* is found, the *current_hash* is correct, as it is very difficult to obtain the same hash from two different sets of data.

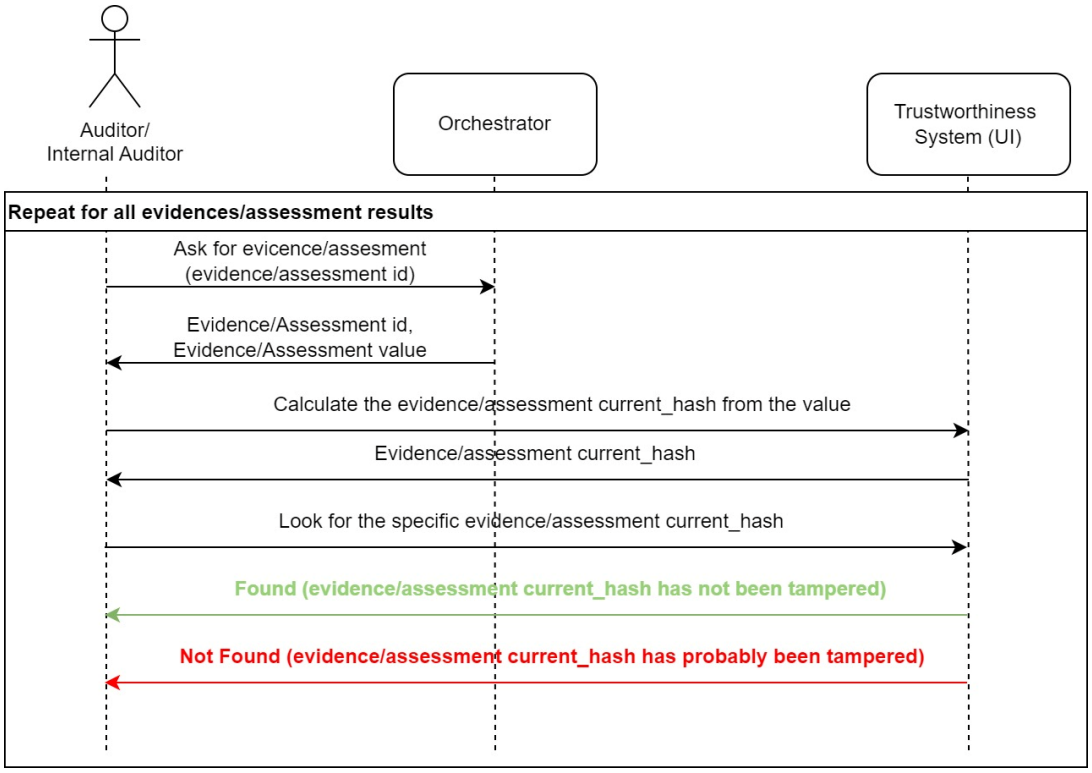

Figure 11. Manual verification using the Orchestrator and MEDINA Evidence Trustworthiness Management System GUI

The main advantage of this solution is that the *Orchestrator* would not need to be updated, not extending its functionality, and maintaining the trustworthiness verification completely on the *MEDINA Evidence Trustworthiness Management System*.

The main disadvantage is that the *MEDINA Evidence Trustworthiness Management System* would need to be updated to be able to obtain the *current_hash*es values. Furthermore, it could be risky to provide evidence/assessment result values to the *MEDINA Evidence Trustworthiness Management System* user interface, which is not locally deployed and is offered as a service from TECNALIA (sensitive data should not leave its local premises). Finally, it is a manual process for the auditors.

- Instead of using the evidence/assessment results values from the *Orchestrator*, it could be possible to obtain them directly from the MEDINA evidence storage, which needs to be publicly accessible by the auditors. Once auditors obtain the values, they follow the same process as explained before, following the steps described in Figure 12.

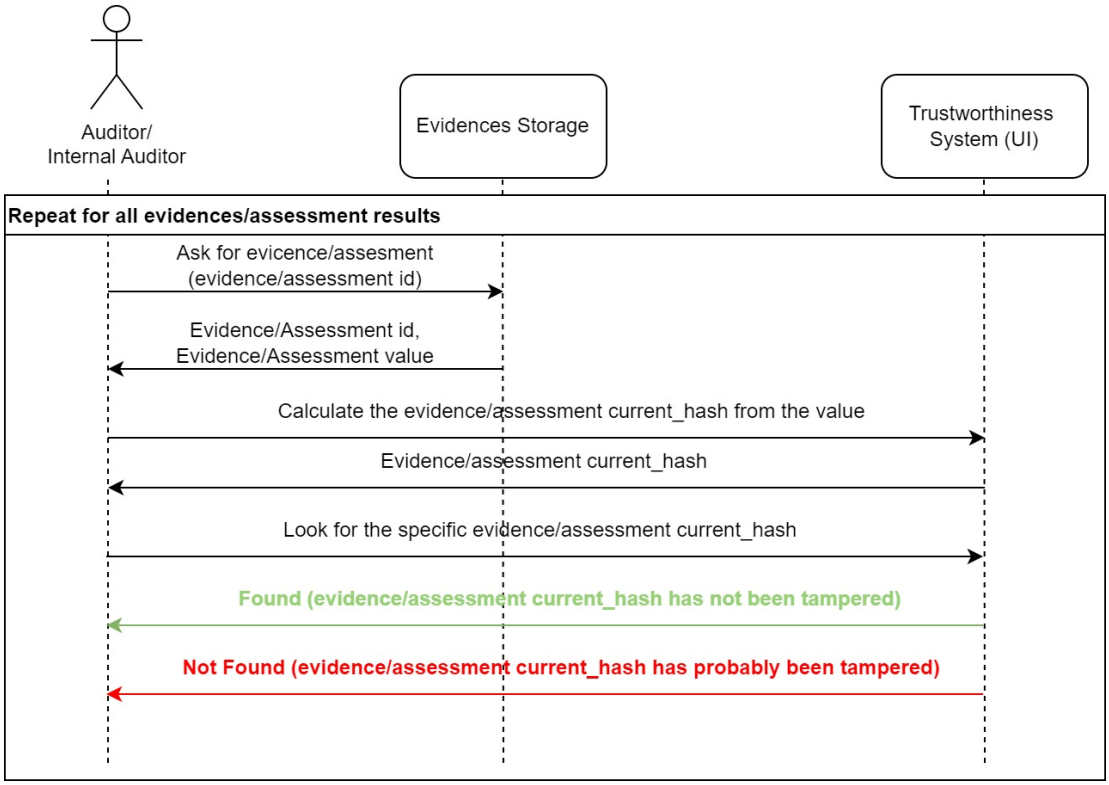

Figure 12. Manual verification via the evidence storage and MEDINA Evidence Trustworthiness Management System GUI

The main advantage of this solution is that the *Orchestrator* is not involved, not extending its functionality, and maintaining the trustworthiness verification completely on the *MEDINA Evidence Trustworthiness Management System*.

The main disadvantage is that the *MEDINA Evidence Trustworthiness Management System* would need to be updated to be able to obtain the *current_hashes* values. Furthermore, it could be risky to provide evidence/assessment result values to the *MEDINA Evidence Trustworthiness Management System* user interface, which is not locally deployed and is offered as a service from TECNALIA (sensitive data should not leave its local premises). Besides, the evidence storage needs to be publicly accessible by auditors. Finally, it is a manual process for the auditors.

5.4.3 Calculation of Hashes in an Additional Service

A new MEDINA component could be designed for the evidence and assessment results verification in order to avoid modifications to the *Orchestrator* and/or the *MEDINA Evidence Trustworthiness Management System***.** There are several options:

- Once the current evidence/assessment results values are obtained from the *Orchestrator* or the evidence storage, the auditors will use an additional service for the verification process. This additional service would automatically obtain the *current_hash* values and verify them on the Blockchain through the *MEDINA Evidence Trustworthiness Management System* API. As a result, a TRUE/FALSE result would be shown to the auditors as shown in Figure 13.

  The main advantage is that the *Orchestrator* and the *MEDINA Evidence Trustworthiness Management System* do not need any update and there is no mix on functionalities. On the contrary, an additional service would be needed inside the MEDINA framework for the

verification functionality (it could be also provided as a service from outside, but this would mean a risk due to the need of the sensitive data to leave the local premises).

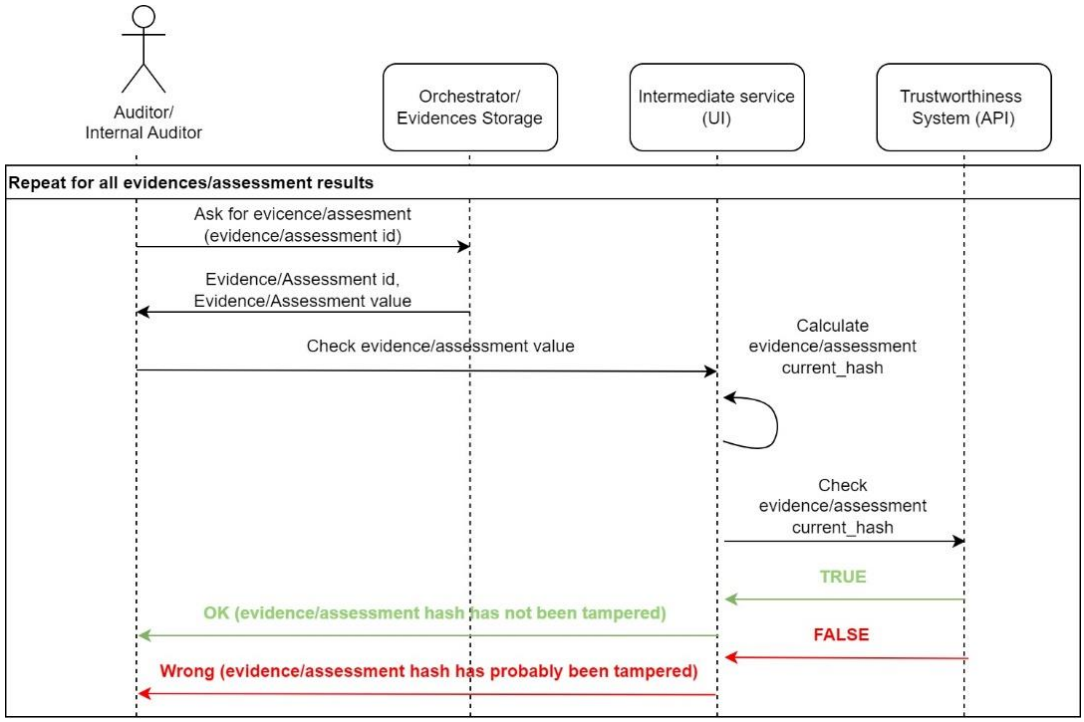

Figure 13. Automatic verification using an intermediate additional service

- The additional service could be the entry point for the evidence/assessment result verification process, providing a user interface and the hash obtaining the hash verification functionalities through the *Orchestrator*/evidence storage and the *MEDINA Evidence Trustworthiness Management System* API (see Figure 14).

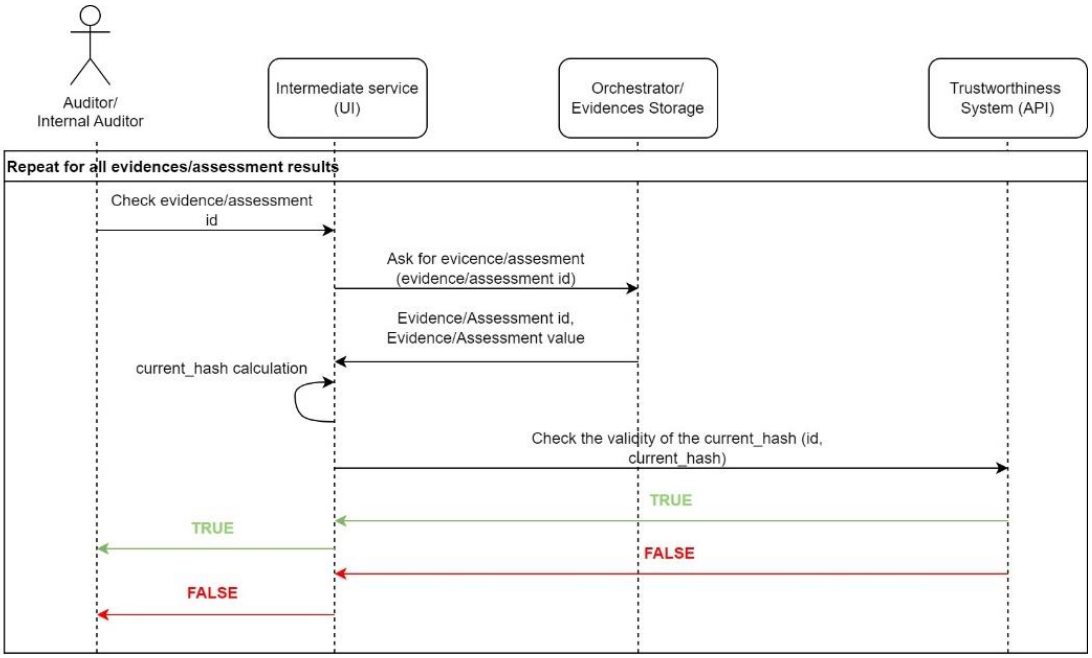

Figure 14. Automatic verification using an additional service as entry point

The main advantage is that the *Orchestrator* and the *MEDINA Evidence Trustworthiness Management System* do not need any update and there is no mix on functionalities. Besides, just one user interface will be used by the auditors. On the contrary, an additional service would be needed inside the MEDINA framework for the verification functionality (it could be also provided as a service from outside, but this would mean a risk due to the need of the sensitive data to leave the local premises).

5.5 Summary and Future Work

5.5.1 Summary

In this section, we have presented an extensive theoretical analysis required for the MEDINA *MEDINA Evidence Trustworthiness Management System*, including existing risks, mitigative technologies, and with a special focus on hash algorithms needed as a guarantee of data integrity. Furthermore, different workflows for the evidence and assessment results verification have been analysed.

5.5.2 Limitations and Future Work

One limitation is that secure storage alternatives considered in the analysis have been limited to traditional databases, replicated databases and Blockchain-based solutions. Although Blockchain is demonstrated to be suitable, there could be some concerns about its energy consumption, cost, or scalability.

Taking this into account, future work includes the following tasks:

- Consumption analysis of Blockchain technology in general, and Quorum technology in particular.
- Cost analysis of Blockchain technology in general, and Quorum technology in particular.
- Scalability analysis of the Quorum technology.
- Additional theoretical studies will be carried out based on the *MEDINA Evidence Trustworthiness Management System* implementation (Task 3.4) and the theoretical analysis needs that will be identified.

6 MEDINA Automation of the Cloud Security Certification Life-Cycle

After evaluating the assessment results, i.e., aggregating and weighing them (see Section 4), a decision needs to be made about how these results should influence the state of the respective certificate. This management of certificates is done by the *Life-Cycle Manager* (*LCM*). Please note that the Risk Assessment (Task 4.4) processes the results of the certification evaluation before forwarding them to the *Life-Cycle Manager* (see also deliverable D4.4 [1]).

This section describes the MEDINA approach to managing the certificate life-cycle and consists of four main parts: a summary of risks associated to the automatic management of certificates, the description of the *Life-Cycle Manager*, the description of the *SSI Framework*, and a discussion of how the two components address the previously identified risks.

6.1 Risks and Mitigations in the MEDINA Certification Management

Certificate management ensures that certificates reflect the current security level of a cloud service by translating evaluation results into a certificate state, and possibly making that state public. There are various risks that threaten this activity, and different possibilities to counter these risks.

6.1.1 Potential Risks

In traditional certification approaches, issued certificates are published and often can be verified with the certification authority. In this case a (potential) customer may, e.g., use the certification body's website to see if the auditee's name is listed there.

Reputation damage: One potential risk in certificate management concerns the auditee's reputation which can significantly be impacted by the evaluation results, which an automated certification process continuously generates. If, for instance, a component or data flow is manipulated to modify the outcome of the evaluation of assessment results, a competitor may damage a cloud service provider's reputation. At the same time, a malicious auditee may also try to manipulate the logic of this evaluation process to generate compliant results that ultimately result in the desired certificate state. The publication of a certificate's state — or state change — therefore needs to be protected from intentional and unintentional interference.

Denial of service: Also, certificate management needs to ensure that the current state of any certificate is available to be viewed (and verified) by stakeholders, e.g., in a public registry. If a certificate is not available, it is not possible to fully trust the claimed security of the respective auditee. Usually, however, the verification of a certificate is not time-critical, so a temporary non-availability of a certificate is neither very likely, nor is it very harmful.

Loss of trust: A further, more abstract risk is the loss of trust that is put into the certification process and its actors. The certificate's value highly depends on that trust — an erroneous certificate state change could therefore also severely hurt the trust into the continuous certification process, and the certification, itself.

Summarizing, the protection goals that are relevant are the following.

- Confidentiality of evaluation details, such as non-compliances of specific resources (only the certificate state is public)
- Integrity of the certificate state
- Availability of the certificate

Attack vectors towards these goals and assets are therefore as follows:

1. **Modify the logic of the certificate management component:** A malicious attacker may try to modify the certificate manager to generate non-compliant results, e.g., to hurt competitors.
2. **Forge a certificate:** An attacker may try to create an illegitimate certificate that is trusted by potential customers.
3. **Delete a certificate:** An attacker may try to delete an existing certificate, e.g., to hurt a competitor.
4. **Deny the retrieval of a certificate:** Using a denial-of-service attack, an attacker may try to prevent that the existence or state of the certificate can be retraced.
5. **Disclose sensitive certificate details:** An attacker may disclose details about the state of a certificate, e.g., non-compliance details of a suspended certificate, possibly revealing vulnerabilities of the CSP.

As described above, the impacts can include reputation damage to the auditee, but also reputation damage to the certification process, the certificate, and the certification authority.

6.1.2 Discussion of Smart Contracts as a Possible Mitigation

One possibility to protect the integrity of the certification management logic (attack vector 1) is to use *smart contracts*. Ante [30] defines smart contracts as "*decentrally anchored scripts on blockchains or similar infrastructures that allow the transparent execution of predefined processes*". Historically, the term *smart contract* has not necessarily been associated with blockchains. For example, Röscheisen et al. [31] described a smart contract already in 1998 as a "*digital representation of an agreement between two or more parties" that has "a structured […] interface, code that implements behaviour, state (e.g. the validity status, the number of times a right was exercised, etc.), and a set of textual descriptions*".

In the documentation of Ethereum, the most popular platform for the deployment of blockchain-based smart contracts, a smart contract is defined as "*a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain*"[2]. Most cryptocurrencies, e.g., Bitcoin, use a blockchain to store transactions between accounts. To make the execution of smart contracts possible, Ethereum also stores code and data on the Blockchain to enable the execution of the Ethereum Virtual Machine.

It is furthermore important to note that there is a difference between a smart contract and a *legal* contract: a smart contract, e.g., anchored on a public blockchain, does not necessarily represent a legally binding document[3].

The goal of using smart contracts is usually the elimination of trusted third parties. One reason is that trusted third parties may sometimes not be fully trusted by all stakeholders. Also, they incur additional cost and overhead into a transaction. For example, certification audits may consume many person days to prepare documentation, conduct interviews, create reports, etc. The most prominent examples of avoiding trusted third parties are cryptocurrencies, like Bitcoin and Ethereum, which aim at eliminating the need for financial institutions to manage a currency and accounts.

In the traditional certification process, trust is mainly established via the trusted certification authority – a reputable third party that has no interest in issuing an undeserved certificate. In

[2] https://ethereum.org/en/developers/docs/smart-contracts/
[3] Please note that the authors are no legal experts, but analyse the possibility of using smart contracts merely from a technical perspective.

the continuous process, in contrast, trust needs to be established through a reliable design and technical implementation that guarantee the correct management of certificates.

In the following, some inherent risks of using smart contracts are described:

- A risk of using smart contracts is that they could be deployed including bugs and vulnerabilities, which may only be discovered after their deployment. While various approaches have been proposed to validate a smart contract's purpose and to eliminate bugs before their deployment, this risk can never be fully eliminated.
- Also, the environmental impact of blockchain technologies should be considered. Storing a large number of transactions can, depending on the algorithm, result in high amounts of energy consumption.
- A further considerable disadvantage of smart contracts is that there is no possibility for remediation or consideration if the contract fails. Traditional contracts often include a severability clause which may define that the purpose of the contract is still effective even though a part of it emerges to be unenforceable. This way, the general purpose of the contract can be upheld. In smart contracts, in contrast, there is no room for interpretation or consideration. In the context of certification, this means that in case a part of the contract becomes, e.g., outdated, illegal, or unenforceable, there is no possibility to change its scope or logic.

The topic of using smart contracts for different purposes has also been discussed in the literature. Some works have investigated, for example, how to transform business processes to smart contract-based processes that eliminate intermediaries and work more efficiently, e.g., proposing frameworks [32] and compilation processes [33].

Few works actually investigate the challenges that occur and that have to be solved to port business processes to the Blockchain. For example, Carminati et al. [34] identify challenges for the application of smart contracts for inter-organizational business processes. As such, their results are largely applicable to cloud certification as well, which is a business process between several organizations, possibly including a CAB, an auditee, and one or more cloud vendors.

They identify five challenges which are discussed in the context of certification as follows:

- **Data integrity:** Important data that is processed by smart contracts has to be integrity-protected as well. Smart contracts should therefore store all relevant data in protected transactions. In the context of certification, this challenge also raises the question of input integrity. For example, a smart contract may obtain input data from a cloud service, e.g., about encryption configurations. If this data, however, is maliciously modified then they will be used in the unalterable logic of the smart contract which in turn produces outcomes that are stored unchangeably on the Blockchain.
- **Data confidentiality:** While it is a current research problem to allow for confidential blockchain transactions, it is not a standard feature. The data that a smart contract generates and uses are therefore public – assuming that a public Blockchain is used. When making certification decisions, this public information could potentially reveal sensitive information about the CSP's security problems.
- **Confidentiality of the process:** Carminati et al. [34] also raise the issue of confidentiality of smart contracts themselves, i.e., their program logic, since the process flow itself may reveal sensitive information. When implementing certification processes, this is not a relevant issue, since the workflow of a certification process, as well as its decision criteria, are usually defined in public documents.
- **Trust in the correct execution of the process:** Process trust has to be established for all participants in the business process. One threat to this trust is the possible data

breaches and tampering attacks that may happen when the smart contract interacts with off-chain components which are not integrity-protected. This is also a major issue for implementing certificates as smart contracts since the certification process itself requires trust by customers in this process. In traditional certification approaches, this trust is established through the auditors who represent a trusted third party.

- **Data provenance:** Data provenance refers to the origin of data and its "history". In MEDINA, there is an inherent trust assumption for the tools that gather and assess evidence, so this data's provenance is assumed to be verified. In future work, however, the issue of making the provenance of evidence that is, e.g., gathered from a public cloud provider, should be addressed.

In summary, smart contracts can be used to reliably execute a piece of code, e.g., for translating evaluation results into a certificate state according to pre-defined criteria. However, elements of the certification pipeline, i.e., all systems and tools that contribute to the continuous certification including evidence gathering, evaluation, and certificate management, that are not (or cannot be) deployed in an integrity-protected environment, such as a Blockchain, can severely limit the usefulness of a Blockchain-deployed smart contract. For example, APIs for the evidence gathering may change, configurations for the smart contract may change (e.g., the service location or scope), or the requirements for publishing or managing the requirements may change (e.g., the states and their conditions). Also, the data transmission from off-chain elements to the smart contract may be attacked. As soon as such a condition changes, the smart contract may become non-functional, and the continuous certification process may be interrupted.

The question therefore is whether these conditions can be assumed to remain unchanged, and whether a smart contract can mitigate the previously identified risks, e.g., the risk of malicious modification of the certificate management logic. On the one hand, smart contracts can reliably protect the integrity and execution of a piece of code. They can therefore be seen as a mitigation for attack vectors 1 and 2 (see Section 6.1.1). On the other hand, this mitigation introduces new risks, e.g., unfixable bugs, disclosure of sensitive information, and the challenge of protecting the integrity of the other parts of the certification pipeline remain.

Due to the additional risks that the usage of smart contract introduces for certificate management, we have decided to handle the risks differently in MEDINA. We review them again in Section 6.4.

6.2 Design and Implementation of the Life-Cycle Manager

As stated above, implementation of the *Life-Cycle Manager* component is independent from smart contracts and blockchains in general. The current second-version of the *Life-Cycle Manager* is published as an open-source project[4].

6.2.1 Functional Requirements

The following requirements, defined in D5.1 [35] and updated in D5.2 [18], are addressed in this second iteration (however, not fully implemented yet).

Requirement id	ACLM.01
Short title	Cloud security certification issuance
Description	Based on the quality evaluation results, the system will push appropriate entities (CAB) to issue and sign security certifications for the cloud providers.

[4] https://git.code.tecnalia.com/medina/public/life-cycle-manager

Status	Partially implemented

Requirement id	**ACLM.02**
Short title	Automatic cloud security certification update
Description	Based on the quality evaluation results, the system will push appropriate entities (CAB) to update the security certifications for the cloud providers.
Status	Partially implemented

Requirement id	**ACLM.03**
Short title	Cloud security certification revocation
Description	Based on quality evaluation results, the system will push appropriate entities (CAB) to revoke the security certifications for the cloud providers.
Status	Partially implemented

Requirement id	**ACLM.04**
Short title	Continuous update of the certificate state
Description	The certificate life-cycle management component must continuously, i.e., in high-frequency intervals, convert the evaluation results from the CCE to the corresponding certificate state.
Status	Partially implemented

Requirement id	**ACLM.06**
Short title	Compliance with EUCS assurance levels and certificate states
Description	The certificate life-cycle management component must map the certificate states and assurance levels defined in the EUCS scheme.
Status	Partially implemented

Requirement id	**ACLM.07**
Short title	Interface for a public registry
Description	The life-cycle management component must provide an interface for publishing the certificate status in a public registry by the corresponding entities (CAB).
Status	Partially implemented

Regarding **ACLM.08** (*The life-cycle management component can be implemented in a smart contract to ensure a tamper-proof execution*), an evaluation of the use of smart contracts has been carried out (see Section 6.1.2). Based on the results, it was decided to discard this requirement.

6.2.2 Certificate States

In the following, we briefly review the certificate states defined by the EUCS:

- **New Certificate** for newly issued certificates, following an assessment with positive outcome.
- **Continued** for certificates that have been reassessed and should not reflect any changes.

- **Renewed** for certificates that have been reassessed and whose validity is extended. Updates to the certificate's information may be added.
- **Updated** for certificates that have been reassessed and which remain valid but need updates in their information.
- **Suspended** for certificates that have been reassessed with the outcome that the service does not conform to the requirements of the targeted assurance level anymore. The state is also entered if a periodic reassessment has not been conducted in due time.
- **Withdrawn** for certificates that have not been maintained after the suspension.

These states and transitions are reflected in the state machine model as shown in Figure 15. The dashed lines refer to the renewal flow where a certificate is first withdrawn and then renewed, e.g., to reflect a different assurance level.

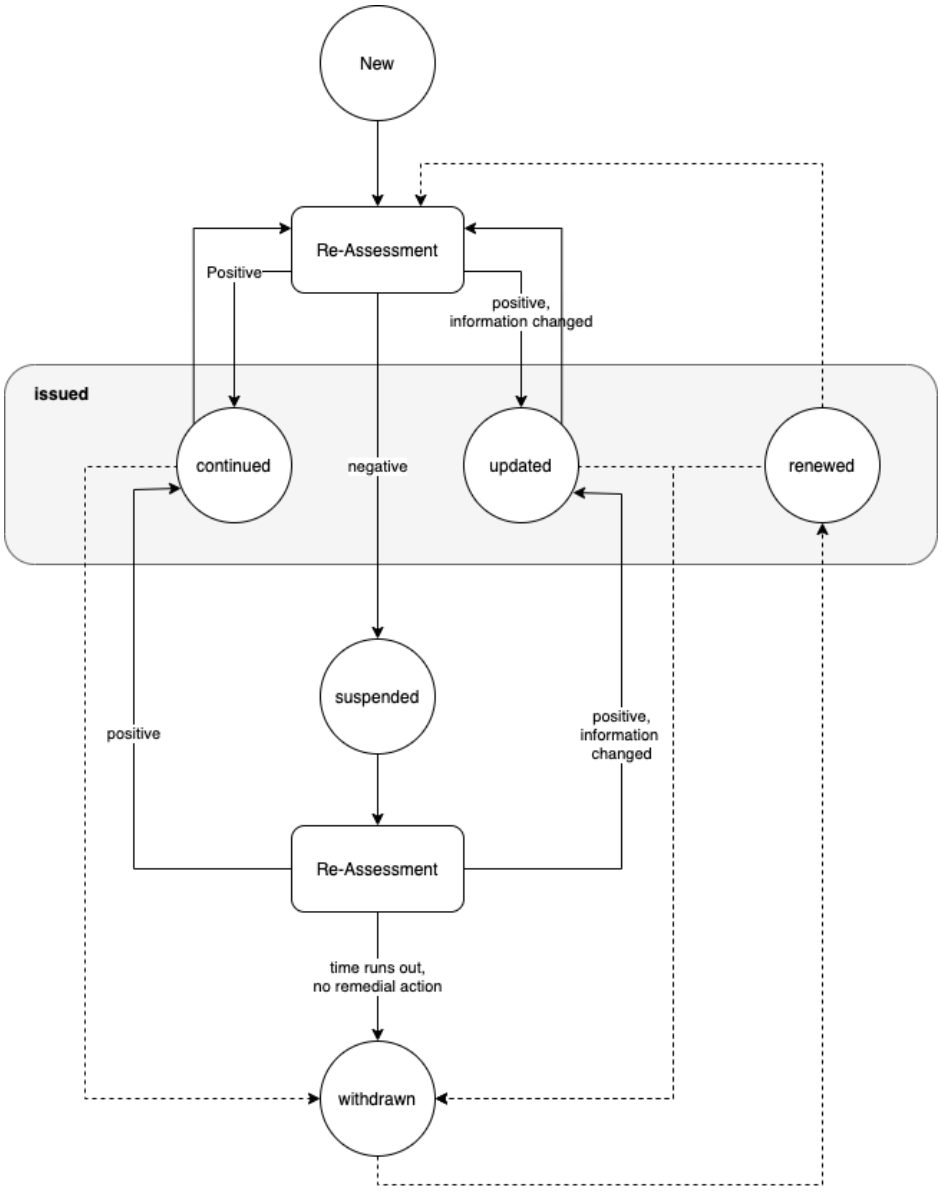

Figure 15. A state machine model of the EUCS phases (source: MEDINA's own contribution)

6.2.3 Automating Certification Decisions

There are multiple possibilities and degrees to which the above-described certificate state changes may be automated. In the following, we discuss several possible options and derive a decision for MEDINA.

6.2.3.1 Option 1: No Automation

The first, and simplest, possibility is to not automate certificate decisions at all. This is the normal case in state-of-the-art audits where auditors make point-in-time audits to manually check documentation, conduct interviews, etc. In this case, the MEDINA framework can still considerably benefit CSPs and auditors as it prepares evidence and assessment results in a way that they can easily be presented in a point-in-time audit.

There are still two reasons that suggest *some* amount of automation: First, new certification frameworks, like the EUCS [12], demand an automatic monitoring of certain security requirements. Second, such an automatic monitoring generates evidence in a high frequency which may overwhelm (internal and external) auditors. Third, meaningful information for the automation of certification decisions is available: Overall risk scores, as well as time rules, and operational effectiveness data, can be used to derive a reasonable decision on if there are significant underlying problems in the system. Also, they can do so very quickly, and therefore improve overall security.

6.2.3.2 Option 2: Complete Automation

The second alternative presents the other extreme: completely automating the certificate updates, especially suspending, withdrawing, and continuing the certificate. This requires the *LCM* to take decisions based on the data mentioned above, e.g., provided by the *Risk Assessment and Optimisation Framework* (RAOF) (see D4.4 [1]).

However, integrating this information into the certificate maintenance decisions is challenging, because meaningful thresholds need to be defined. For instance, a threshold of *50* may be defined for the risk value generated by the RAOF and suspend a certificate when the value is higher than the threshold. Since no general thresholds can be defined for all systems, they should always be validated by an expert, e.g., in the initial audit.

Furthermore, there are more risks in the automation of certification decisions: First, bugs in the system may trigger a high-risk value, resulting in harming the CSP's reputation if the risk is automatically translated into a suspension. Second, the thresholds may be tampered with by unnoticed attackers. Third, there is also a general risk of neglecting important information about the cloud service (in comparison with manual audits) due to focusing on one or few metrics like the overall risk value.

6.2.3.3 Option 3: Automation in Selected Cases

We can furthermore analyse the certificate maintenance cases and decision rules specified in the EUCS [12] to identify cases that are easier to automate and less prone to the risks described above in Option 2. Consider the cases from the EUCS presented in Table 13.

Table 13. Certificate maintenance decisions defined in the EUCS [12]

Case	Nominal Decision
The maintenance evaluation activities have been performed and reviewed, and have determined that the cloud service still fulfils the	Continue the certificate until the next periodic assessment or until its expiration date

requirements without significant changes in the service	
The maintenance evaluation activities have been performed and reviewed, and have determined that the cloud service still fulfils the requirements and the changes impact the security of users without any reduction in the scope of certification or assurance level	Update the certificate with the new information and continue the certificate until the next periodic assessment or until its expiration date
A renewal conformity assessment has been performed and reviewed, and have determined that the cloud service still fulfils the requirements, possibly with changes that impact the security of users without any reduction in the scope of certification or assurance level	Renew the certificate with a new expiration date and if required with the new information
The maintenance evaluation activities have been performed and reviewed, and have determined that the cloud service only fulfils the requirements after reducing the scope of certification or reducing the assurance level	Withdraw the certificate and issue a new certificate with the reduced scope or assurance level, possibly with a different expiration date
The maintenance evaluation activities have been performed and reviewed, have determined that the cloud service does not fulfil the requirements anymore, and action from the CSP is possible to maintain the certificate at the same assurance level and scope, though not immediately, or improper use of the certificate is not solved by suitable retractions and appropriate corrective actions by the CSP.	Suspend the certificate pending remedial action from the CSP
The maintenance evaluation activities have been performed and reviewed, and have determined that the cloud service does not fulfil the requirements anymore	Withdraw the certificate
The periodic assessment has not been performed in due time	Suspend the certificate pending remedial action from the CSP
Remediation action has not been performed in due time after suspension	Withdraw the certificate

Two exemplify the automation potential, consider the two following two cases:

- "The maintenance evaluation activities have been performed and reviewed, have determined that the cloud service does not fulfil the requirements anymore, and action from the CSP is possible to maintain the certificate at the same assurance level and scope, though not immediately, or improper use of the certificate is not solved by suitable retractions and appropriate corrective actions by the CSP."

In the above case, the resulting decision is to (temporarily) suspend the certificate. However, it is difficult to determine what exactly constitutes the criterion that the service does not fulfil the requirements anymore, e.g., which requirements, or how many, and to which degree.

- "The periodic assessment has not been performed in due time."

This second case is simpler: it requires to suspend the certificate if the periodic assessment has not been performed in the required interval. When continuous monitoring is in place, it can easily be determined if a periodic assessment has been conducted or not and the certificate status can be changed automatically.

6.2.3.4 Option 4: Automation Barring Manual Verification

In MEDINA, we aim at exploring the potentials for automation, and at the same time mitigate risks as far as possible. We therefore automate the certification decision making but introduce a manual verification by an auditor before the new certificate state is published.

To this end, we currently combine two types of information to make an automated certificate update, i.e., a risk value and an operational effectiveness value (explained in the Section ?). After making the decision, internal auditors may be alerted automatically to allow for a quick remediation. Additionally, the CAB is informed to allow external auditors to review information about the (potential) certificate change. They can then accept or reject the change.

The advantages of this approach are twofold: First, the CSP benefits from quick information on the current certificate and potential problems in the cloud system, and second, the manual verification ensures that the CSP's reputation is not harmed in case of erroneous automatic decisions.

Yet, this approach may still present disadvantages. Apart from potential problems regarding the validity of the automatic results, it can be the case that internal and external auditors are overwhelmed if the system generates too many changes and alerts. In this case, auditors may even disregard alerts.

6.2.4 Implementation

The current implementation of the *Life-Cycle Manager* (*LCM*) represents the state machine as seen in Figure 15 and is written in the Go programming language.

6.2.4.1 Functional Description

The *Life-Cycle Manager*'s purpose is to automatically make certificate maintenance decisions. At the time of writing, three types of automatic transitions for certificate states are conceptually designed and partly implemented.

Risk value: First, transitions may be triggered based on the risk value reported by the *Risk Assessment and Optimisation Framework*. A configurable threshold value in the RAOF determines whether the risk value is considered a minor or major deviation. Consequently, the *LCM* suspends the respective certificate if a major deviation is reported. Analogously, the certificate is *continued* if no deviation or a minor deviation is reported. Since risk values are reported frequently, they build the basis of the certificate maintenance decisions in MEDINA.

Operational effectiveness: Second, transitions may be triggered based on the operational effectiveness values reported by the *Continuous Certification Evaluation* component. Please note that currently, this is only partly implemented. Please refer to Section 4 for more information on the calculation of the operational effectiveness values. The results of these calculations are reported to the *LCM* where a configurable threshold value again determines if the value is considered a major deviation and the certificate should be suspended. Operational effectiveness is a rather long-term view, for example calculated over a time period of six months. It is therefore computed in larger intervals, e.g., daily. It is important to note that it overwrites decisions made based on the risk value. For example, a continued certificate due to a low-risk value may indicate that the current state of the respective cloud system is overall compliant, but

may be overwritten by a major deviation due to a low operational effectiveness value, which indicates that compliance was too low in the last six months.

Time rules: *Third, transitions may be triggered based on the time rules defined in the* EUCS *[12]* (see Table 13). For instance, if a certificate is in the suspended state and no remedial action has been done, i.e., no change to a minor deviation has been achieved, it is automatically withdrawn after a configurable time period.

The two most frequent and therefore most important decisions are to *continue* and *suspend* the certificate. If it is withdrawn, it cannot be continued automatically, but needs to be issued newly again. The time-based rules are independent from the risk-based and operational effectiveness-based transitions. Please note also that we assume that all configurable parameters, like thresholds, are reviewed by auditors in the initial audit.

With the logic described in this section, we achieve an automated certificate maintenance which incorporates simple rules, a risk perspective, and a temporal view on the certificate state. This makes the WP4 components an extendible certificate management toolkit that is both sophisticated and easily understandable.

6.2.4.2 Fitting Into Overall MEDINA Architecture

The LCM forms the end of the MEDINA certification pipeline (together with the SSI Framework). It receives information about the current risk value from the *Risk Assessment and Optimization Framework*, operational effectiveness data from the *Continuous Certification Evaluation* component, and stores certificate data in the database managed by the Orchestrator.

6.2.5 Technical Description

6.2.5.1 Prototype Architecture

The LCM's architecture follows a simple structure divided into three parts:

1. The *cmd* module provides the bootstrapping functionality that starts the REST server and establishes a connection to the *Orchestrator.*
2. The *cert* module provides internal logic for managing certificates, e.g., creating, modifying, and deleting them. This includes the storage of changes via the Orchestrator.
3. The *rest* module manages the external APIs, e.g., provision of risk values.

6.2.5.2 Components Description

No sub-components are implemented in this component.

6.2.5.3 Technical Specifications

The LCM provides several APIs for the management of certificates and provision of evaluation data:

- HandleEvaluation: a POST API for the RAOF to provide the identified risk value
- HandleCreation: a POST API for creating a new certificate
- HandleInfoUpdate: a PUT API for updating information on a certificate
- HandleDeletion: a DELETE API for deleting a certificate
- GetStateLog: provides data about a certificate's state history.

6.2.6 Delivery and Usage

6.2.6.1 Package Information

The implementation is structured as follows.

- The main method in the *cmd/life-cycle-manager* package creates a sample certificate and starts the REST API.
- Next, the *models* package contains all data models, currently a user model and a certificate model.
- The *rest* package represents the API of the *Life-Cycle Manager*. It listens to different HTTP routes for different commands, e.g., create a new certificate, or handle a deviation. When a respective command is retrieved, it calls the functionalities of the *cert* package (see below).
- The *cert* package contains most of the actual functionalities: The *rest-util* file contains utility functions to execute HTTP and gRPC connections, as well as other functionalities. The *cert* file contains the methods for creating and modifying certificates and their state histories, and contains methods for the interactions with other components, such as retrieving operational effectiveness data form the CCE component.

Please note that the database, which was part of the previous iteration of the *LCM*, has been removed in this iteration. Instead, the *LCM* now uses the *Orchestrator*'s database. This way, we reduce the overall complexity of managing the MEDINA components, as a MEDINA user does not have to manage an additional database. Also, certificate state changes can be viewed directly in the *Orchestrator* UI rather than in a different interface.

6.2.6.2 Installation Instructions and User Manual

Since the LCM is written in Go it can be built with the following command: `go build cmd/life-cycle-manager/life-cycle-manager.go`

It can then be started with the command: `./life-cycle-manager`

Afterwards, it can be queried via the specified APIs, and risk values can be provided via the *Risk Assessment and Optimisation Framework* as specified in D4.4 [1].

6.2.6.3 Licensing Information

The LCM is licensed under the open source Apache License 2.0.

6.2.6.4 Download

The LCM implementation is available in the public MEDINA repository: https://git.code.tecnalia.com/medina/public/life-cycle-manager

6.2.7 Interface for a Public Registry

Currently, the *LCM* reports certificate suspensions and withdrawals to the *SSI Framework*, so the CAB can review the report and decide whether the state change should be published.

ENISA will in the future provide a certification website, which is defined in the Cybersecurity Act (Art. 85). Final certificate changes should therefore be reported to this website. The website or its interface is, however, not yet defined at the time of writing and will further be investigated in the next iteration.

6.3 Self-Sovereign Identity (SSI) Framework

As described in Section 6.2, the automated decision taken by the *LCM* is reported to the CAB if it is a suspension or withdrawal of the certificate. The CAB is represented in the *SSI Framework* as the *issuer* who creates the official certificate, signs it, and issues it to the CSP. This section describes the current state of the *SSI Framework* in detail.

Please note that as this tool is additional to the base MEDINA framework, it is provided as a proof-of-concept, to validate the suitability of using SSI for assisting the CAB operation. The specific deployment of the different services will be completely on TECNALIA premises, allowing the required interaction with the MEDINA framework.

6.3.1 Functional Description

The *Self-Sovereign Identity (SSI) Framework* provides the CSPs with the capability to manage their own security certificates as part of their identity through verifiable credentials. "To manage their own identity" ultimately means that they store their identity on their own "user space" without intervention of a third-party.

The *SSI Framework* is not only composed of the CSP component to store and control the credentials about themselves. It is also composed of the issuer component which provides the CAB a way to issue verifiable credentials about the security certificates related to the CSPs; and the client´s component which provides a way to ask and verify proofs of different security certificates features. In this sense, privacy is an important requirement within MEDINA, as several security certificates features are considered sensitive and must be treated carefully. The *SSI Framework* is capable of sharing sensitive information in a confidential way by keeping user´s identity out of third parties, that act as identity silos, reducing the risk of identity theft; but also, by using Zero-Knowledge Proofs (ZKPs). ZKPs preserve user´s privacy using cryptography to proof that a CSP has some attributes without disclosing these attributes.

Figure 16 shows the main components of the SSI-based verifiable cloud security certification architecture. The different services for the different actors and their relationships have been identified.

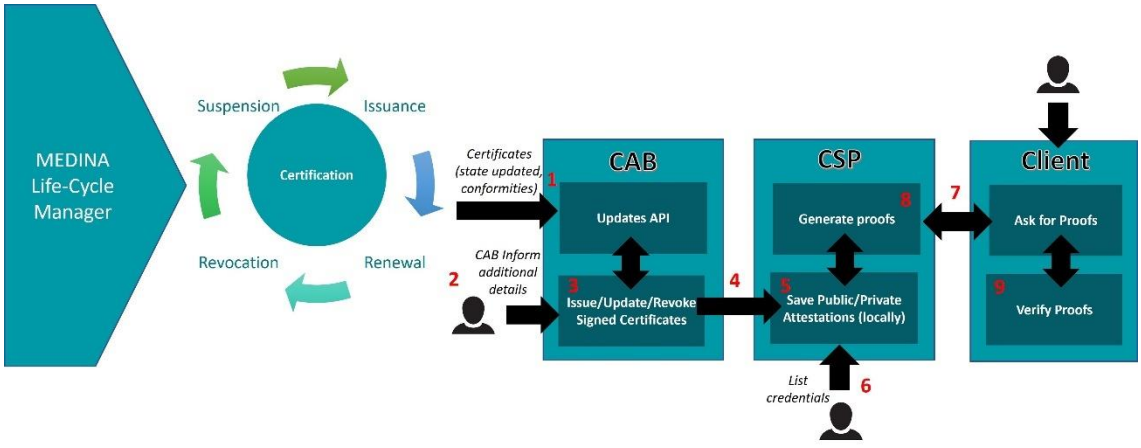

Figure 16. MEDINA SSI-based verifiable cloud security certification functional architecture (source: MEDINA's own contribution)

Below is the collection of requirements (described in D5.2 [18]) related to the *SSI Framework* and a description of how and to what extent these requirements are implemented at this point of development.

Requirement id	SSI.01
Short title	Cloud security certificate issuance
Description	The system should provide a way for appropriate entities (CAB) to issue and sign security certifications for the cloud providers as indicated by the automated certificate *Life-Cycle Manager*.
Status	Partially implemented
Comments	The functionality is already implemented but the *SSI Framework* has not been integrated with the *Life-Cycle Manager* yet.

Requirement id	SSI.02
Short title	Cloud security certificate update
Description	The system should provide a way for appropriate entities (CAB) to update security certifications for the cloud providers as indicated by the *Life-Cycle Manager*.
Status	Partially implemented
Comments	The functionality is already implemented but the *SSI Framework* has not been integrated with the *Life-Cycle Manager* yet.

Requirement id	SSI.03
Short title	Cloud security certificate revocation
Description	The system should provide a way for appropriate entities (CAB) to revoke security certifications for the cloud providers as indicated by the *Life-Cycle Manager*.
Status	Partially implemented
Comments	The functionality is already implemented but the *SSI Framework* has not been integrated with the *Life-Cycle Manager* yet.

Requirement id	SSI.04
Short title	Cloud security certificates listing
Description	The system must list the historical cloud security certificates issued, updated and revoked.
Status	Fully Implemented
Comments	The *SSI Framework* allows the CSP to list the owned security certificates issued by the CAB through the SSI-webapp.

Requirement id	SSI.05
Short title	Cloud security certificate verifiable public proofs generation
Description	The system must generate verifiable proofs of the security certificate state on request.
Status	Fully Implemented
Comments	The *SSI Framework* allows the CSP to generate proofs of its security certificates on demand (by a client/customer) through the SSI-webapp.

Requirement id	SSI.06
Short title	Cloud security certificate confidential proofs generation
Description	The system should generate verifiable confidential proofs of the security certificate private parameters on request.
Status	Fully Implemented

Comments	The *SSI Framework* allows the CSP to generate confidential proofs of its security certificates on demand (by a client/customer) by means of ZKPs through the SSI-webapp.

Requirement **id**	**SSI.07**
Short title	Cloud security certificate proofs request and verification
Description	The system should provide a way for appropriate entities (potential clients) to request and verify proofs of the security certificates to the cloud service providers.
Status	Fully Implemented
Comments	The *SSI Framework* allows the clients/customers to request proofs of CSP security certificates attributes through the SSI-webapp.

6.3.1.1 Fitting into overall MEDINA Architecture

The ***SSI Framework*** fits the overall MEDINA architecture, as shown in Figure 17, providing a way to the CAB to issue, update or remove security certificates of CSP using MEDINA Framework. Considering the MEDINA components, only the ***Life-Cycle Manager*** will provide the information needed by the CAB to update the security certificates status.

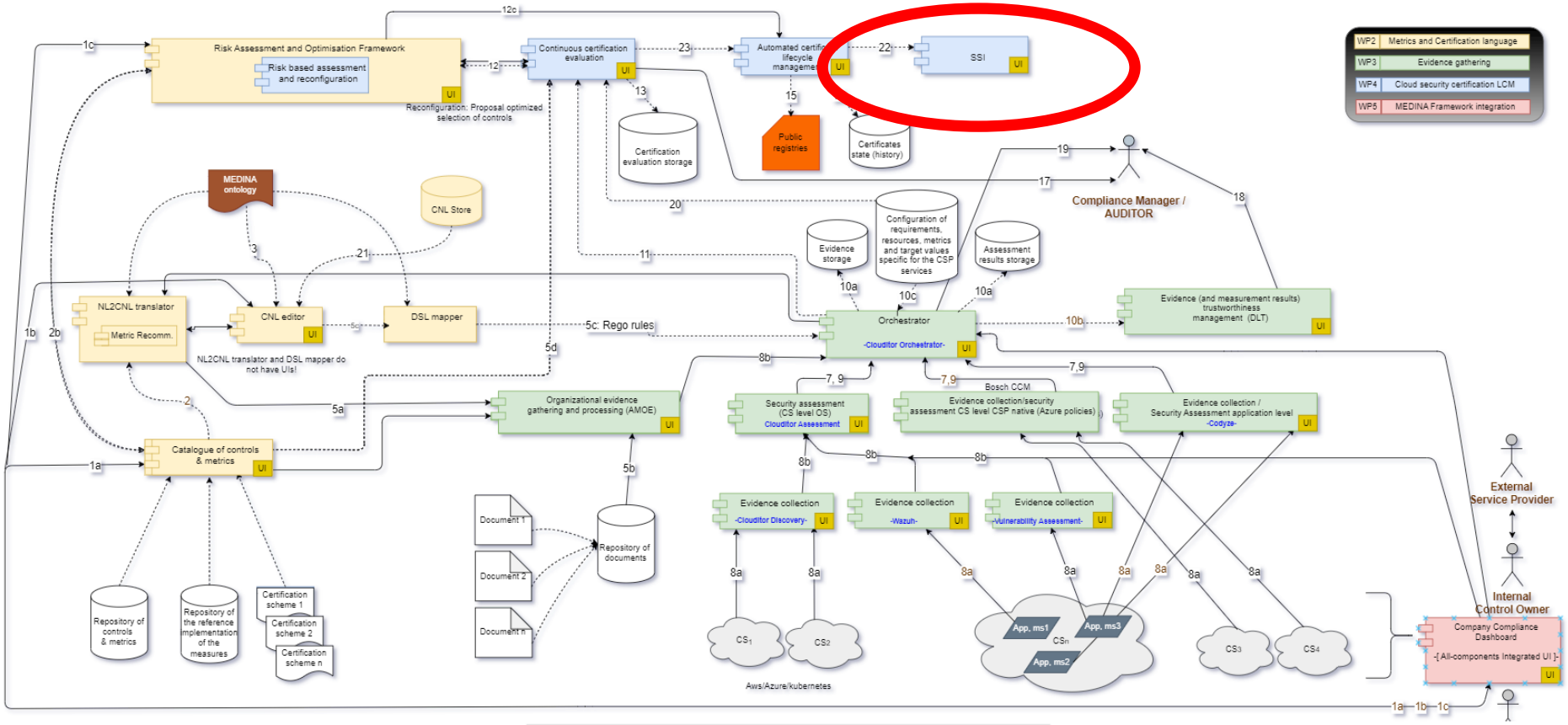

Figure 17. Overall MEDINA Architecture (source: D5.2 [18])

6.3.2 Technical Description

6.3.2.1 Prototype Architecture and Workflow

Figure 18 shows the SSI-based verifiable cloud security certification technical architecture showing its main components: the Aries agents for the issuer, holder, and verifier (SSI-agents); the web controller the three roles (SSI-webapp); and the "updates API" needed by the integration with the *Life-Cycle Manager* (SSI-API). Additionally, an SSI Blockchain network is needed for secure storing the information needed for the secure signatures of the verifiable credentials and proofs (SSI-network).

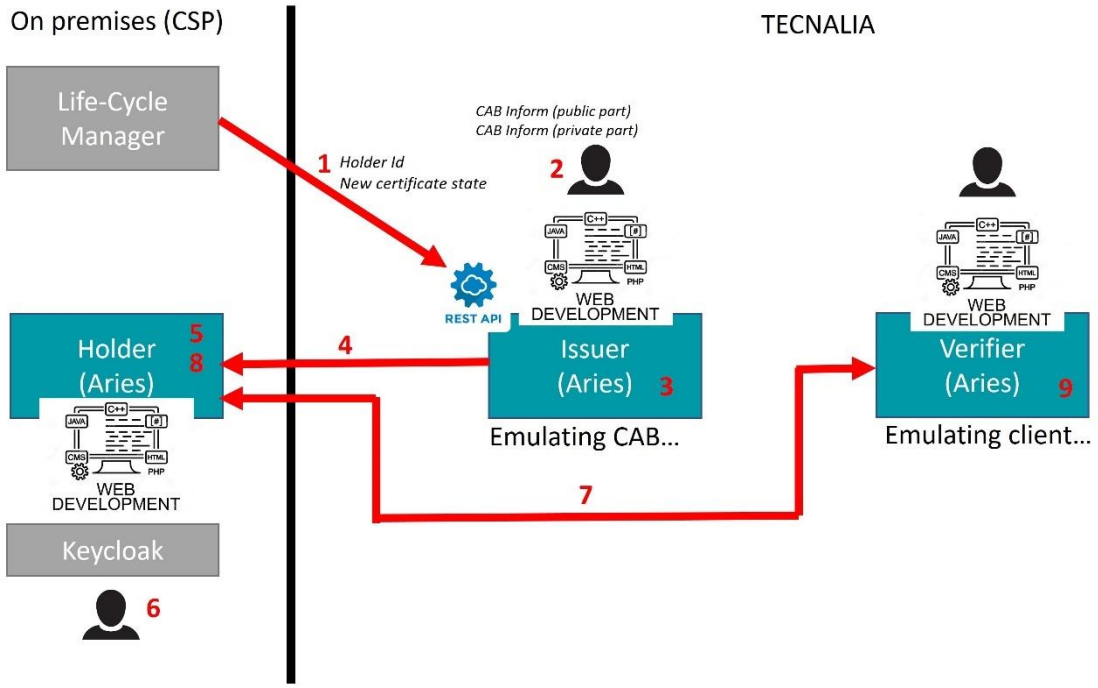

Figure 18. MEDINA SSI-based verifiable cloud security certification technical architecture (source: MEDINA's own contribution)

The holder instance will be deployed on-premises in each CSP as part of the MEDINA framework. However, the issuer and verifier instances will be provided as a service from TECNALIA as a proof-of-concept of the complete solution. The workflow is as follows:

1. The certificate state has changed according to the *Life-Cycle Manager*; this MEDINA component will notify the CAB about this update through the "updates API".
2. The credential with the previous certificate state is automatically revoked and a new credential with the new certificate state is automatically generated for the CAB to validate.
3. When the CAB accesses through the web interface, pending credentials are shown; the CAB will make internal checks and validations, add the CAB report (public and/or private parts) and validate or not the new certificate state.
4. If validated, the new credential with the new certificate state signed by the CAB is issued to the CSP.
5. This new credential is locally stored in the holder (CSP).
6. The CSP staff will be able to access and visualize the credentials they have, at any time, through the web interface.
7. If a client wants to know the current certificate state, he/she will ask the holder for a proof signed by the CAB.

8. When the CSP staff accesses, they will have a list of pending requests, which they will answer as they consider (giving the proof or not).
9. The client will be able to verify the verifiable proof from the CSP in order to probe its authenticity.

6.3.2.2 Description of Components

In this section all components of the *SSI Framework* are described. The *SSI Framework* is composed by 4 parts: the SSI-API, the SSI-network, the SSI-agents, and the SSI-webapp.

SSI-API

The SSI-API connects the *Life-Cycle Manager* with the *SSI Framework*. It is therefore an intermediate middleware. It receives the certificate state updates from MEDINA framework to notify the CAB.

This component is deployed at TECNALIA premises as proof-of-concept, emulating the CAB.

SSI-network

The SSI-network implements the Verifiable Data Registry within the *SSI Framework*, which stores the public cryptographic material, such as: public keys, Decentralized Identifiers (DIDs) and associated metadata. The SSI network acts as a trusted decentralized source with which the different parties can interact.

This component is deployed at TECNALIA premises as a proof-of-concept, emulating a potential Blockchain network of auditors (outside the scope of MEDINA).

SSI-agents

Actors interacting with the *SSI Framework* need specific software that allows them to communicate with each other. This software is known as “agent”. MEDINA users have cloud agents, which means that they are installed in the cloud instead of on the user’s side. Therefore, users call these agents using HTTP methods. Each agent is in charge of keeping the user´s wallet and all the information that this wallet contains, such as verifiable credentials.

SSI-webapp

The SSI-webapp eases the use of the functionality provided by the SSI-agents to the end user. In other words, the user can:

- Connect to one of the available SSI-agents in MEDINA: “issuer”, “holder 1”, “holder 2” and “verifier”. Two holder instances have been deployed, one per use-case.
- Check the current connection status, configuration, and historic usage statistics.
- Manage connections with other SSI-agents: list current connections and create new ones. To create a new connection, the initiator must create an invitation and pass it to the other party. The other user must enter the invitation received to definitely establish the connection. The invitation must be shared with the other user using a secure communication mechanism (e.g., email). If both parties are physically located in the same place, the invitee can scan a QR code generated by the inviter’s browser to comfortably read the invitation.
- List and create DIDs.
- List and create data models.
- Claim ownership of data models and list those owned schemas.
- Create credentials based on those owned schemas for another user and list them.

- Present proofs to a verifier based on credentials the user stores in the wallet.

6.3.2.3 Technical specification

This section includes the programming language, libraries, databases, application servers and other elements required for the implementation of the prototype.

SSI-API

The SSI-API has been developed using Python as programming language. The SSI-API uses Flask, which is a python library for developing HTTP Rest APIs. The API es protected by an API KEY that the certificate *Life-Cycle Manager* needs to use to interact with it.

The SSI-API allows the submission of certificates state updates to the SSI Framework. It exposes a HTTP REST API with the following swagger interface as shown in Figure 19.

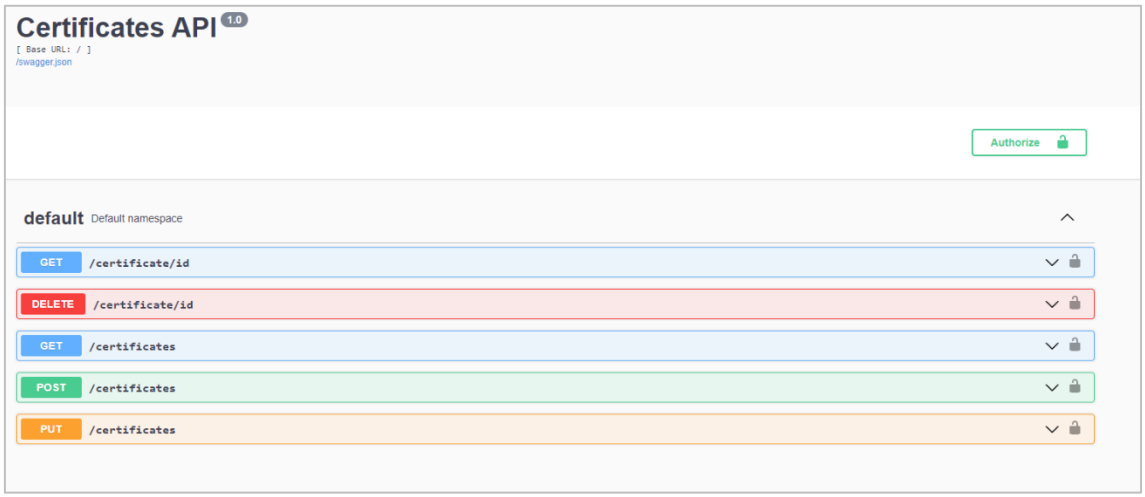

Figure 19. MEDINA SSI-API overview

As Figure 19 shows, the SSI-API es quite simple and is merely a GET/SET API that sends and gets certificate states from the certificate *Life-Cycle Manager* and to the SSI Framework. For each certificate, the fields that the API is processing are:

- Certificate_id: uniquely identifies a certificate.
- Certificate_status: it defined the current certificate state: issuance, renewal, revocation, suspension.

The MEDINA SSI-API is available at: https://api.ssi.medina.bclab.dev/

The following figures show the different endpoints of the SSI-API, showing the required parameters and the possible responses.

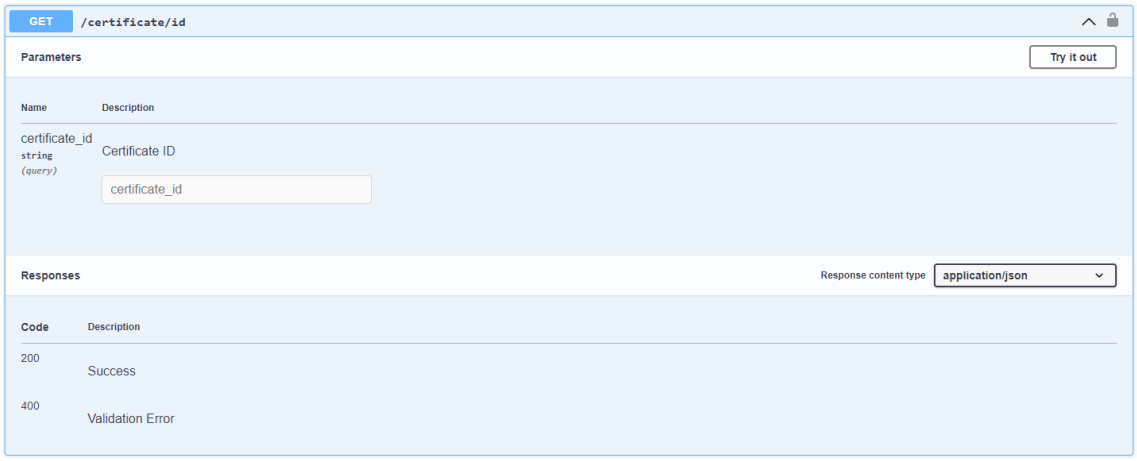

Figure 20. MEDINA SSI-API: GET a Certificate from its certificate_id

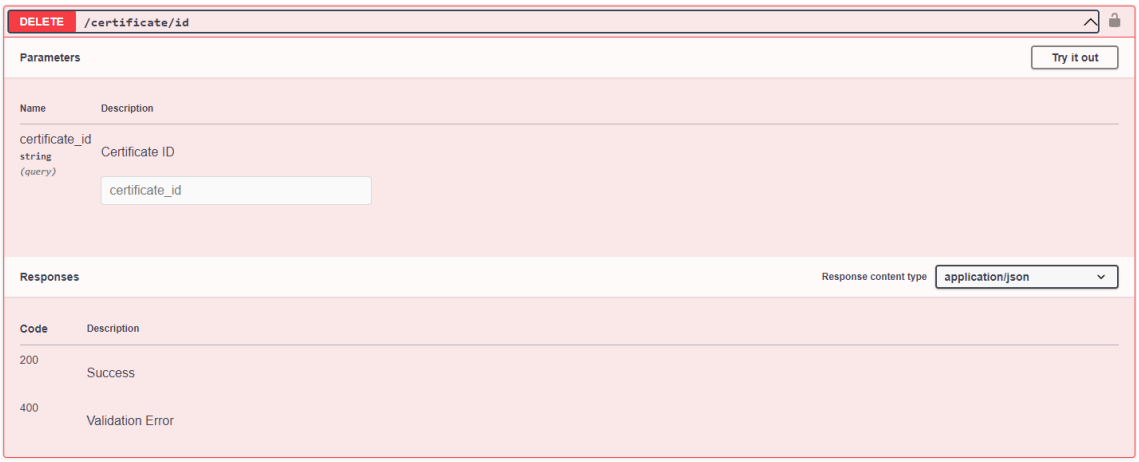

Figure 21. MEDINA SSI-API: DELETE a Certificate from its certificate_id

Figure 22. MEDINA SSI-API: GET all certificates

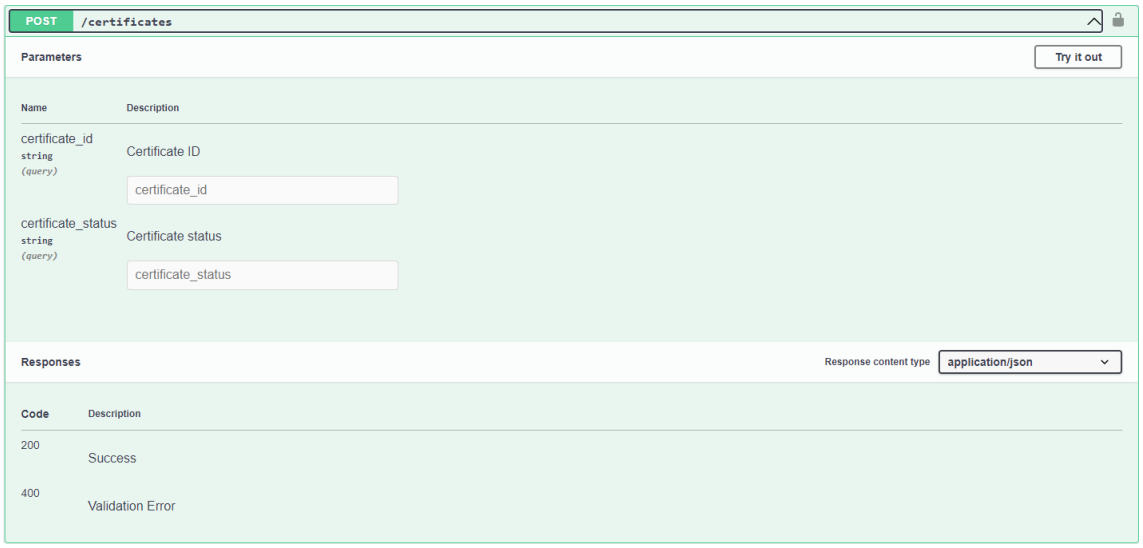

Figure 23. MEDINA SSI-API: POST a Certificate

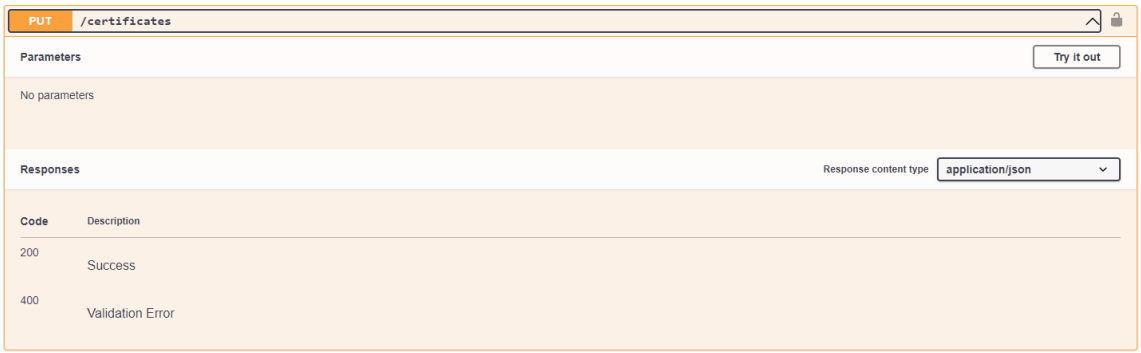

Figure 24. MEDINA SSI-API: UPDATE a Certificate

SSI-network

The SSI network has been developed using Hyperledger Indy. Hyperledger Indy provides a set of utilities that allows the deployment of a Verifiable Data Registry, which has been named SSI-network. Each node can be deployed using a Docker container. It is also possible to define how many virtual or physical machines will have the network.

For MEDINA, a single physical machine has been used with 4 docker containers, each one containing a Hyperledger Indy node. This is considered enough as proof-of-concept; however, it is a basic setup that can be easily escalated to more physical machines if required.

SSI-agents

SSI agents have been implemented using Hyperledger Aries. Hyperledger Aries is a set of libraries that allows the implementation of SSI agents. Each user within the SSI framework has its own cloud agent. The Hyperledger Aries Cloud Agent (aca-py) library has been used to implement the agents. Internally, the SSI agents include an SQLite Database for keeping internal information within the wallet. Once it has been deployed, the agent exposes an HTTP Rest API for allowing interaction with it.

Within the scope of MEDINA project, three agents have been deployed:

- **Issuer:** issues verifiable credentials to the holder associates to the security certificates states. Issuer can be found at: https://issuer.admin.ssi.medina.bclab.dev/api/doc.

- **Holder:** keeps the verifiable credentials in its own wallet and creates verifiable presentations based on these credentials to proof he fulfils certain conditions to the verifier. The holders can be found at:
  - https://holder1.admin.ssi.medina.bclab.dev/api/doc and
  - https://holder2.admin.ssi.medina.bclab.dev/api/doc.
- **Verifier**: verify if the verifiable presentation sent by the holder is correct or not. The verifier can be found at: https://verifier.admin.ssi.medina.bclab.dev/api/doc.

SSI-agents are deployed using a yaml file as will be described in the next section.

SSI-webapp

The web application is a SPA (Single Page-Application) developed using the React framework. It can be deployed in a standard web server as static files. It uses the "Material UI" library for the graphic components in order to keep a clean look and feel. The application is responsive so its UI will adapt to different screen sizes making it appropriate both for normal machines and for mobile devices. When the user goes to the web application for the first time, it will be automatically redirected to the connection page as shown in Figure 25. In this page, the user can select one of the available SSI-agents. After connecting to one of them, the user will be able to use any of the other functionalities described in the following sections.

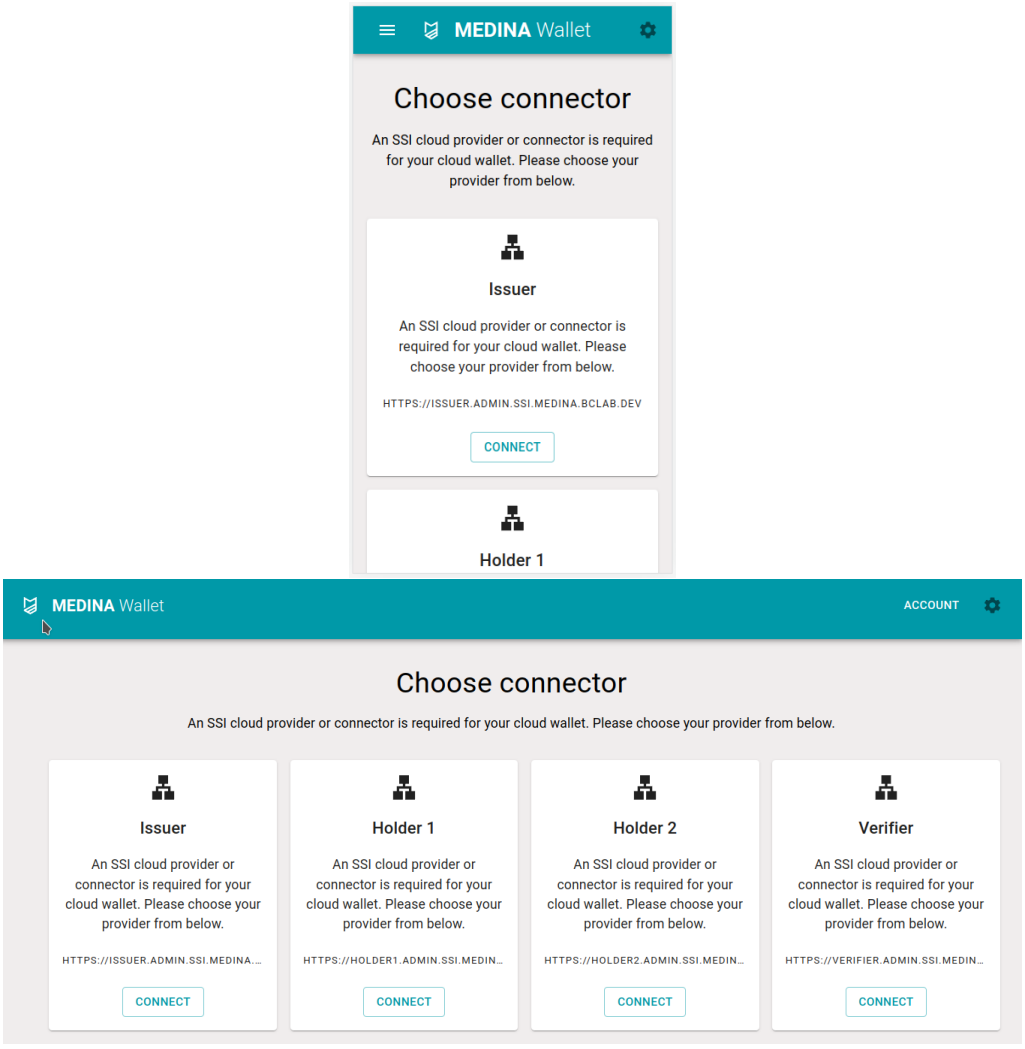

Figure 25. MEDINA SSI-webapp: Connection page visualized in an iPhone SE and in a Desktop browser

6.3.3 Delivery and usage

6.3.3.1 Package information

This section is not applicable in this current version as the whole components are provided as a service from TECNALIA for demo purposes.

6.3.3.2 Installation instructions

This section is not applicable in this current version as the whole components are provided as a service from TECNALIA for demo purposes and no installation is needed by users.

In the next version, the holder instance will need to be installed at the CSP premises as part of the MEDINA framework.

6.3.3.3 User Manual

The manual focuses on the use of the web application (SSI-webapp), as this is the way users will use the SSI framework.

- The first thing the webapp demands to the user is to connect to one of the available SSI-agents.
- The communication between SSI-agents is handled through invitations. It is possible to create a new invitation and share it a providing a QR code with the invitation that other parties can scan to comfortably enter the invitation. The invitation will be then automatically accepted.
- New DID (identifiers), data models or schemas can be defined at any time. This information is needed for issuing credentials.
- Issuer: New credentials can be issued selecting the schema (and corresponding data model) and providing the required details associated to the attributes from the selected schema.
- Holder: The received credentials can be listed at any time.
- Verifier: Proofs for different attributes can be requested.
- Holder: Based on the received credentials, proofs for different attributes can be provided.

The different options are detailed below.

General usage

The first thing the webapp asks to the user to do is to connect to one of the available SSI-agents as shown in Figure 26.

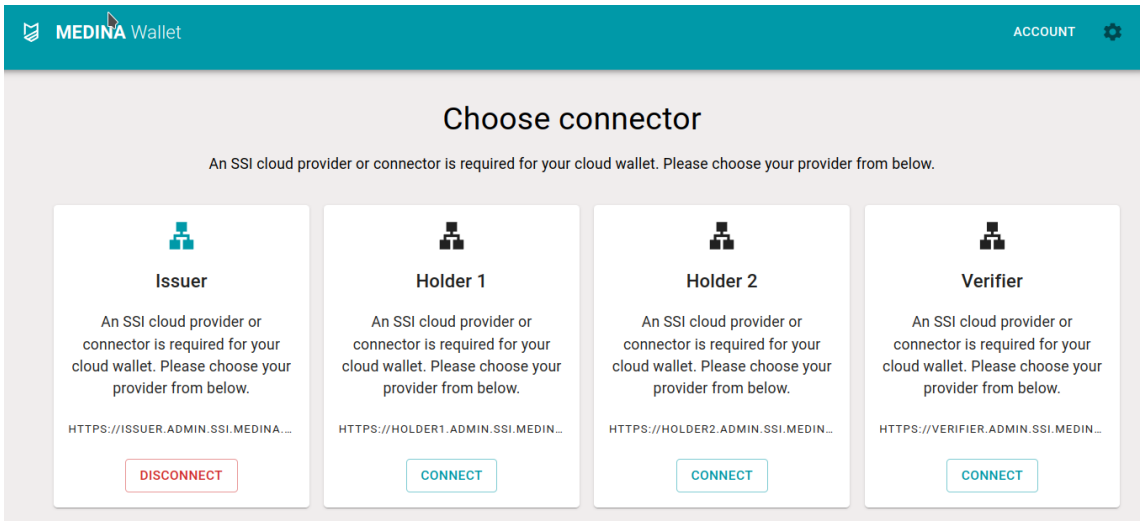

Figure 26. MEDINA SSI-webapp: Connection page while the user is connected to the issuer provider

After selecting one SSI-agent, two new views will become available: status and account pages. The status page (see Figure 27) details some connection stats to verify that everything is working properly, while the account page (see Figure 28) allows the user to access credentials and manage account. This page has six tabs: invitations, DID, data models, owned schemas, credentials, and presentations.

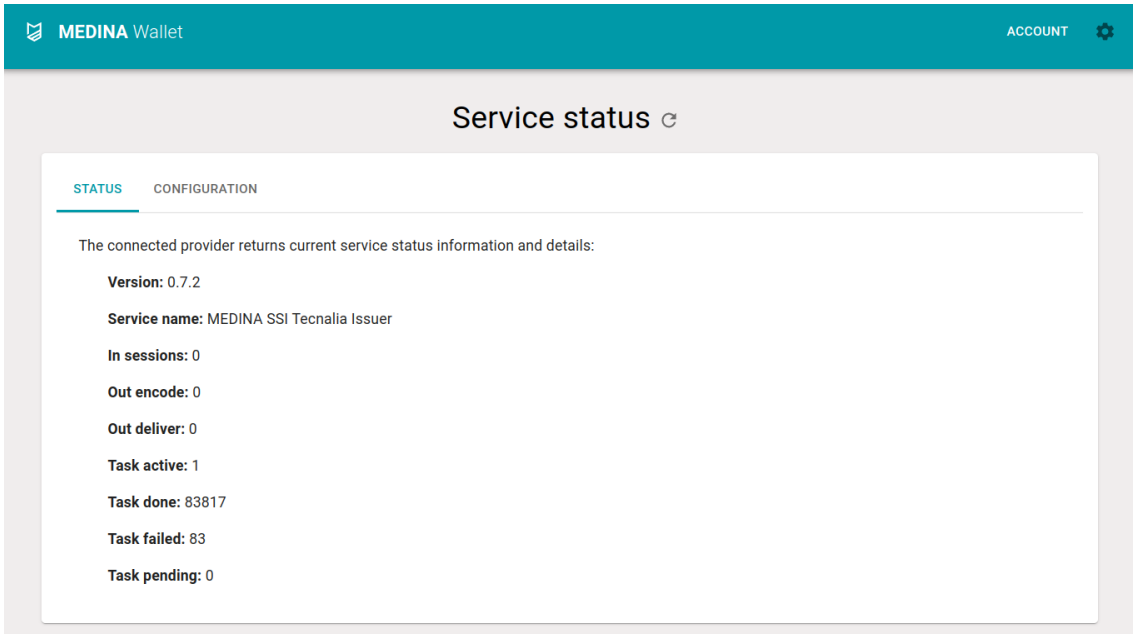

Figure 27. MEDINA SSI-webapp: Web page showing the status of the current connection

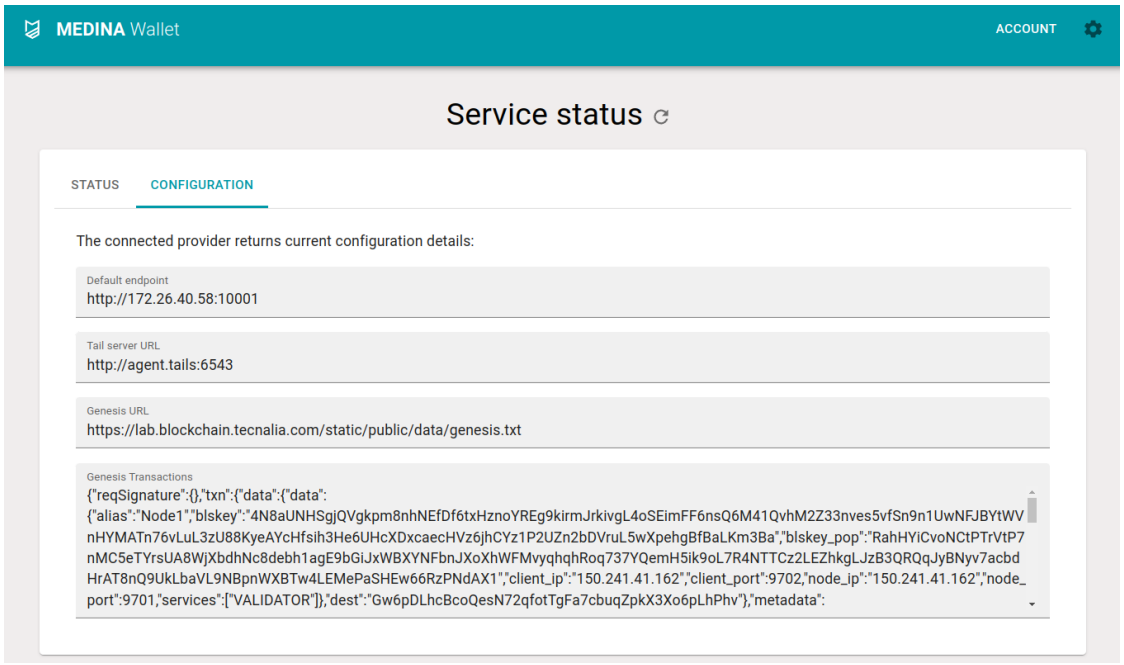

Figure 28. MEDINA SSI-webapp: Connection page showing the configuration of the current connection

Handling invitations

After clicking on the "Invitations" tab, the current connection invitations and their status can be viewed (see Figure 29). Each invitation can be deleted and/or downloaded.

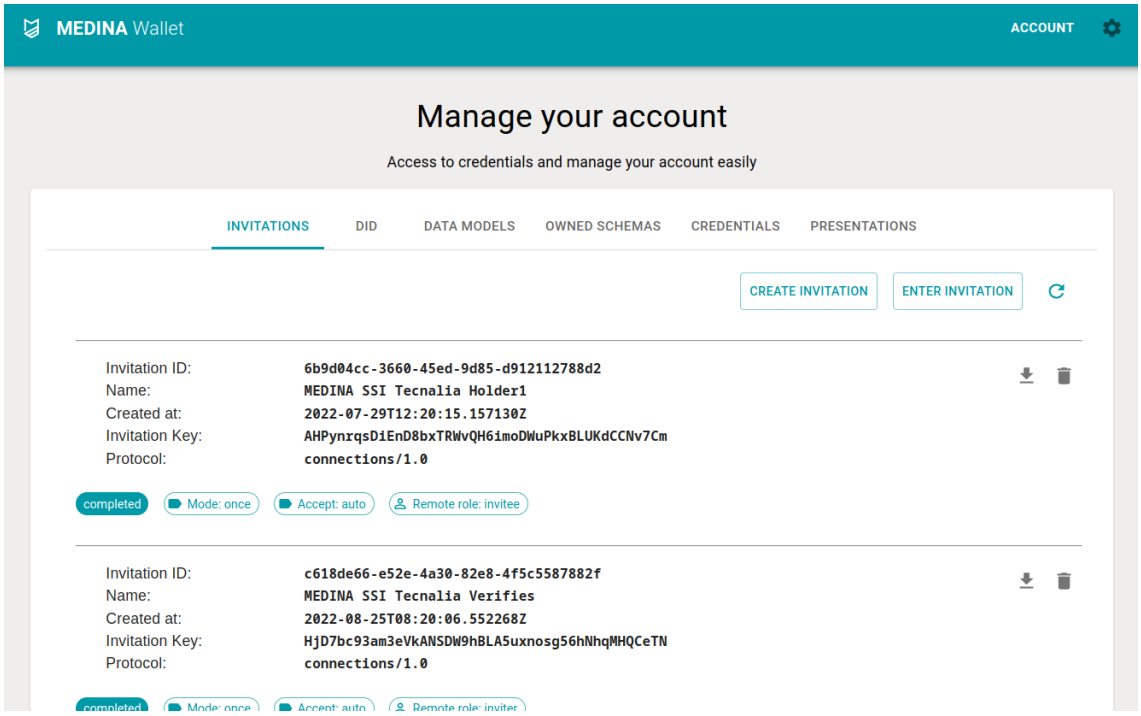

Figure 29. MEDINA SSI-webapp: Invitations tab showing the invitations sent or received by the current user

To create a new invitation, the user must click on the "Create invitation" button as shown in Figure 30. Afterwards, the invitation will be copied to the clipboard and a new entry will appear in the list. This new entry will have a sharing button.

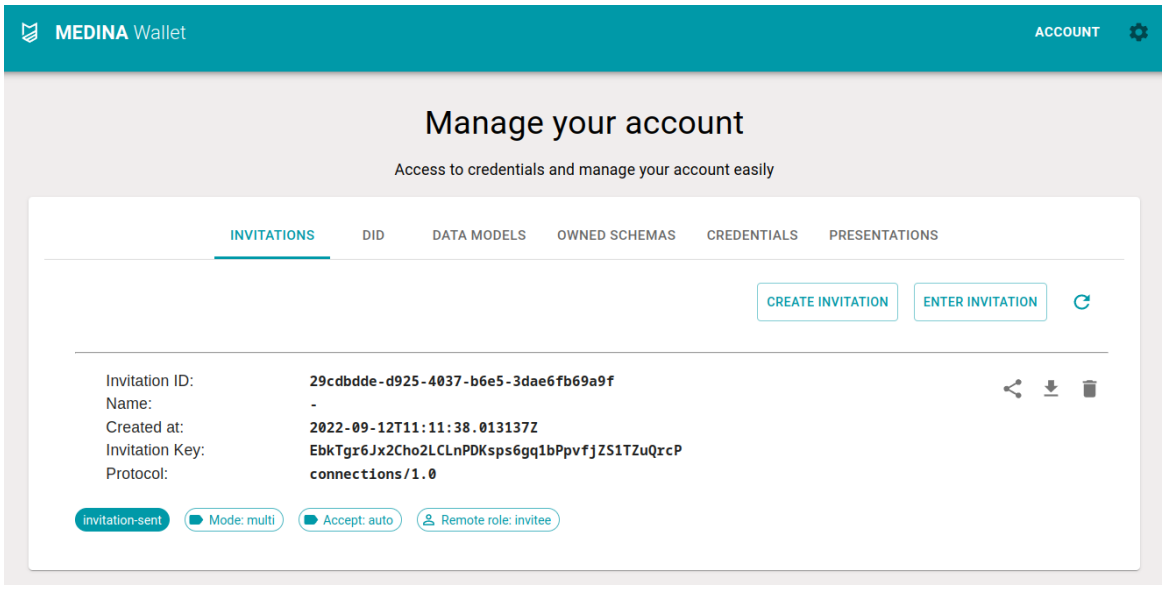

Figure 30. MEDINA SSI-webapp: Invitations tab listing a new invitation to be shared

Clicking on the share button, a dialog will be opened as shown in Figure 31. This dialog contains a QR code with the invitation that the other party can scan to comfortably enter the invitation. Apart from that code, a "Copy to clipboard" button will allow to copy the invitation to the clipboard.

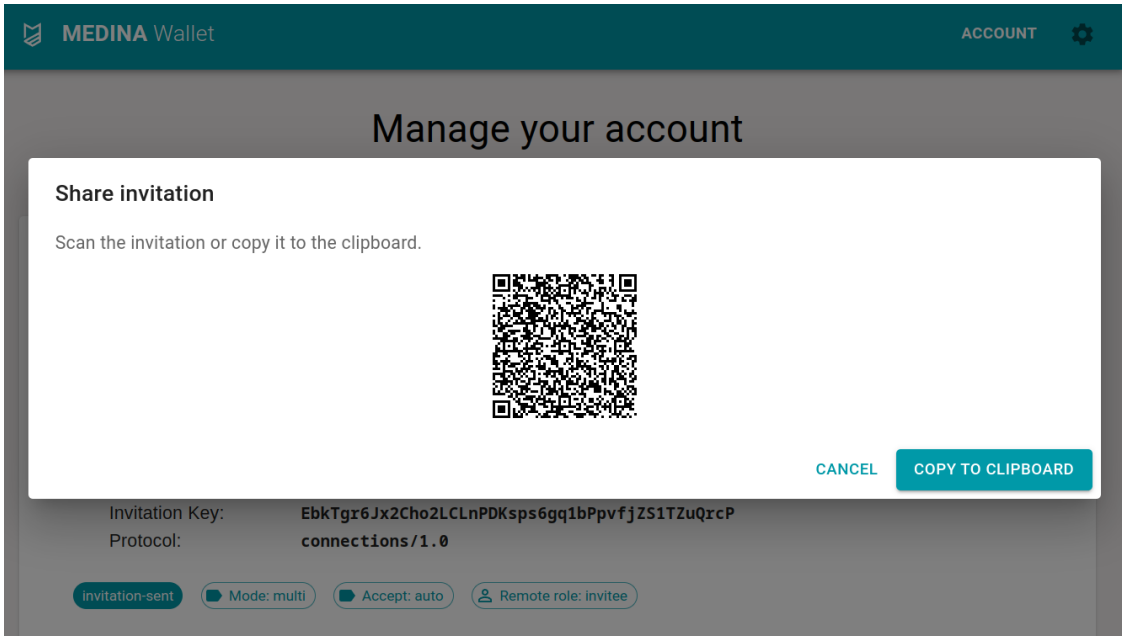

Figure 31. MEDINA SSI-webapp: Dialog to share a connection invitation

The user who will be using this invitation to open a new connection with the former SSI agent must either manually introduce it (see Figure 32), or simply scan it from its browser if both users are in the same location (see Figure 33). The invitation could be shared using any external secure communication mechanism like an email or SMS.

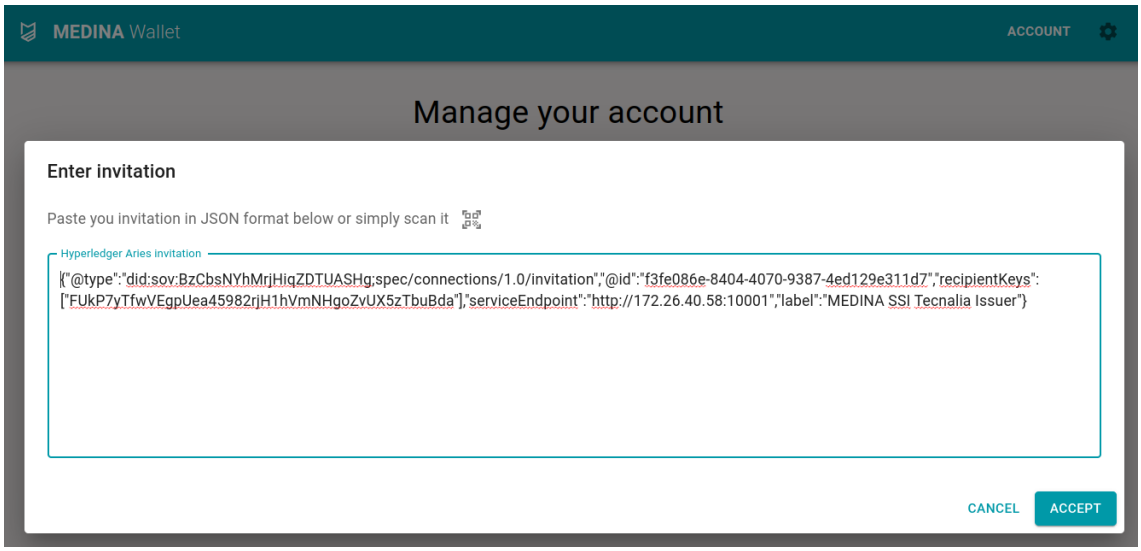

Figure 32. MEDINA SSI-webapp: Dialog used to accept a connection invitation. Form used to manually introduce the invitation

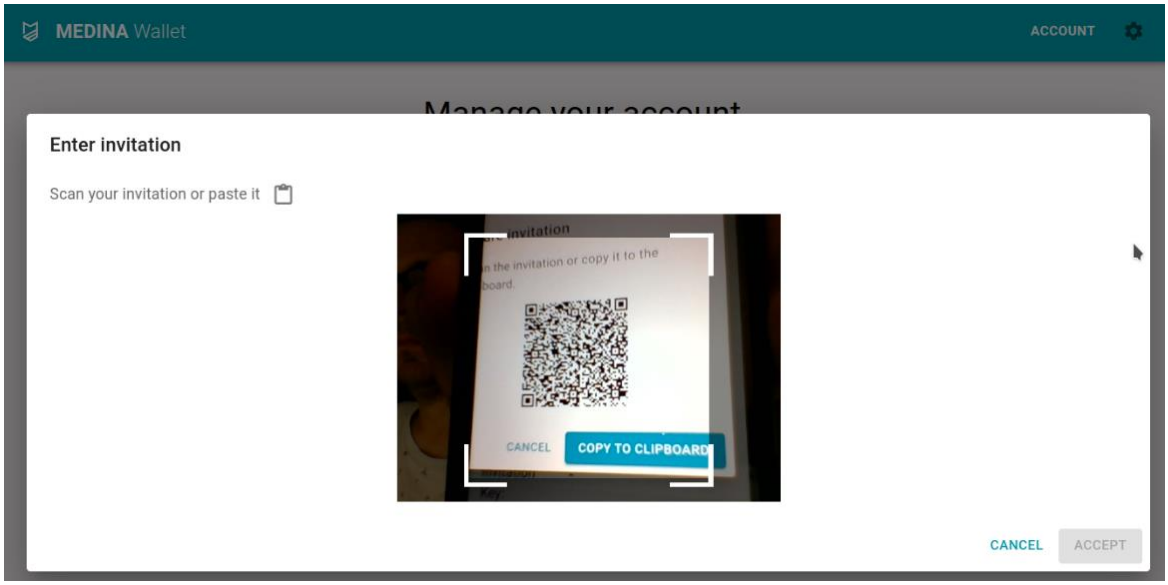

Figure 33. MEDINA SSI-webapp: Dialog used to accept a connection invitation. Scanning mode

Any SSI-agent will automatically accept the invitation and complete the invitation procedure. Eventually both the invitation sender and receiver will see the new connection listed and marked as “completed” (see Figure 34).

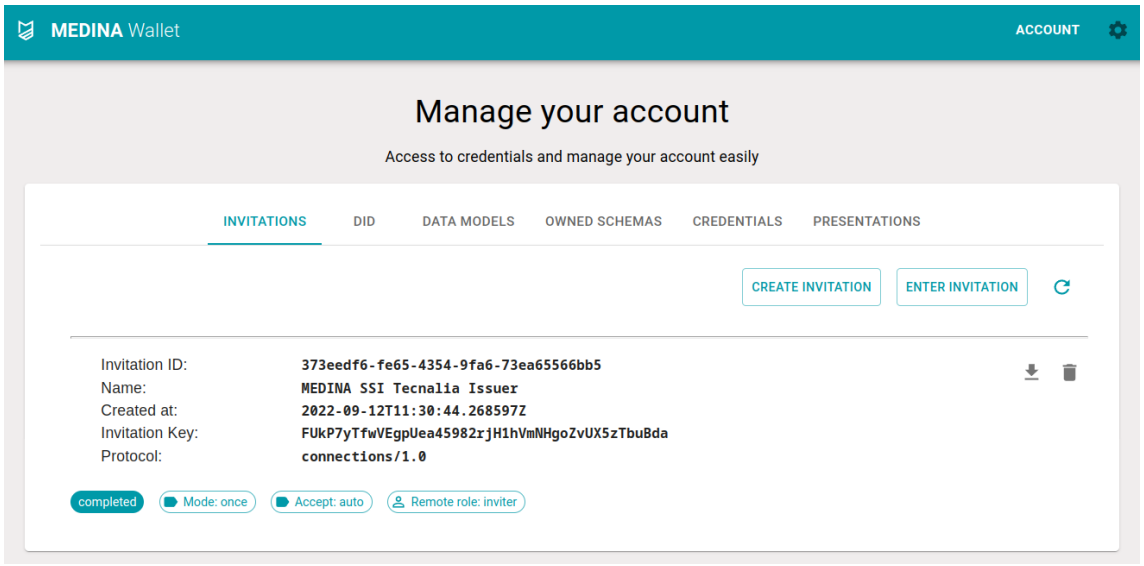

Figure 34. MEDINA SSI-webapp: New invitation marked as completed.

Managing DID, data models and owned schemas

The DID tab lists all the available DIDs (either in wallet or user's public DID) as shown in Figure 35, and allows to create a new DID.

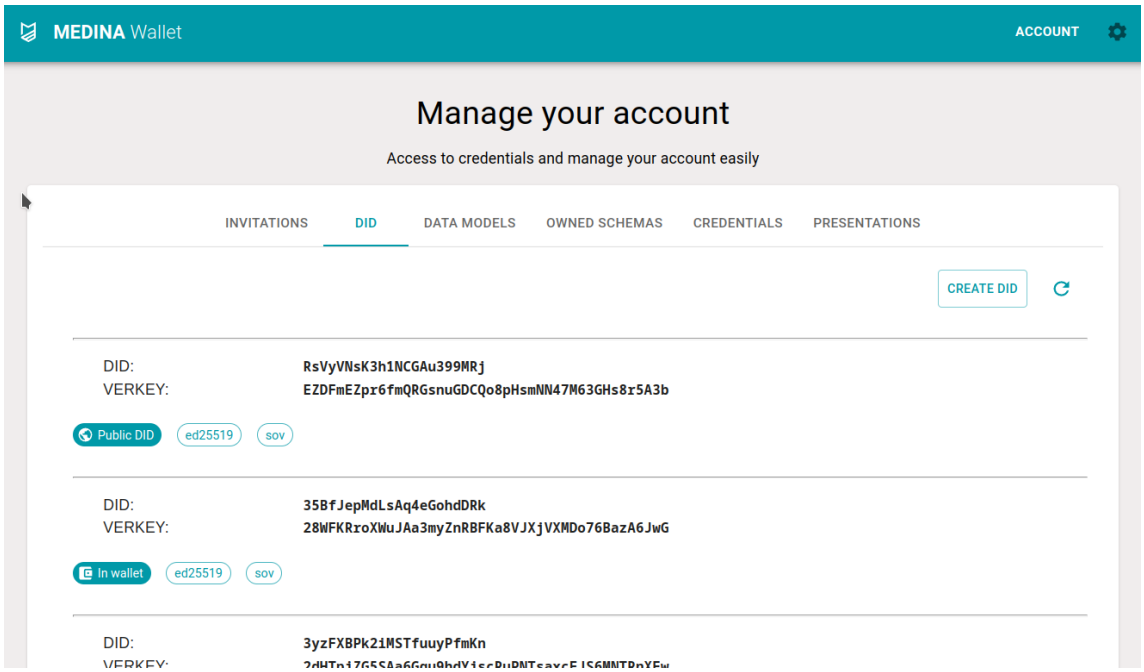

Figure 35. MEDINA SSI-webapp: DID tab showing the DIDs of the current user

The "Data models" tab shows a list with the data models created by the user (see Figure 36). It also allows to create new schemas with the "Create new schema" button (see Figure 37). The model must have a user and a version which univocally identifies it and some attributes (common attributes are provided by the autocompletion field).

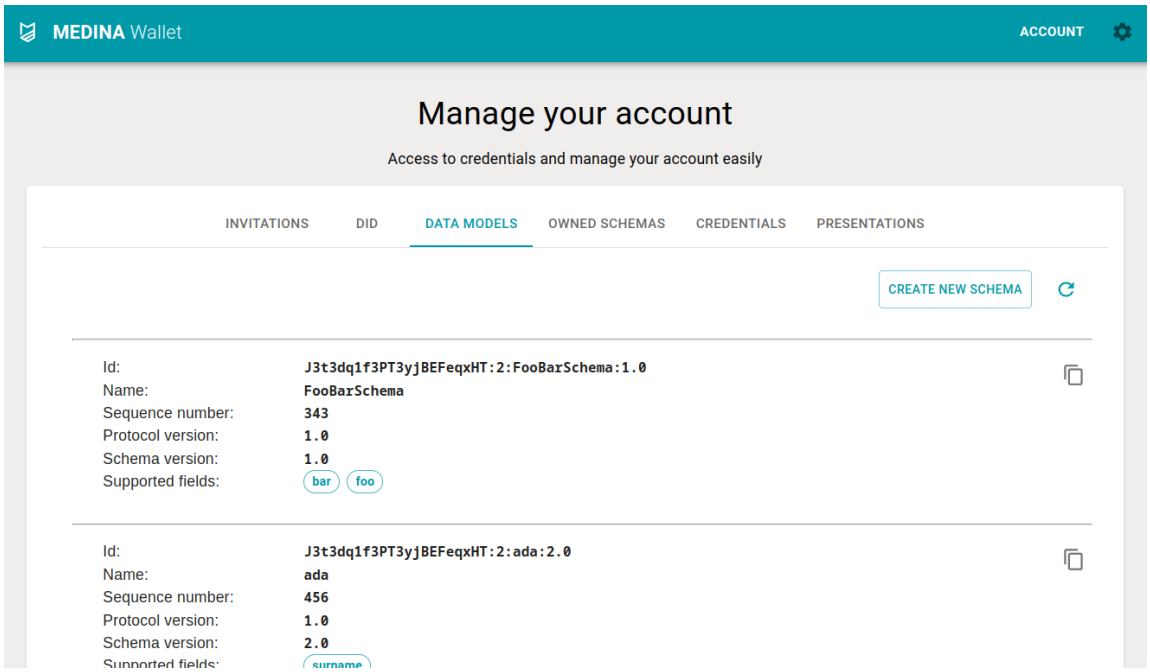

Figure 36. MEDINA SSI-webapp: "Data models" tab listing the details of all the data models

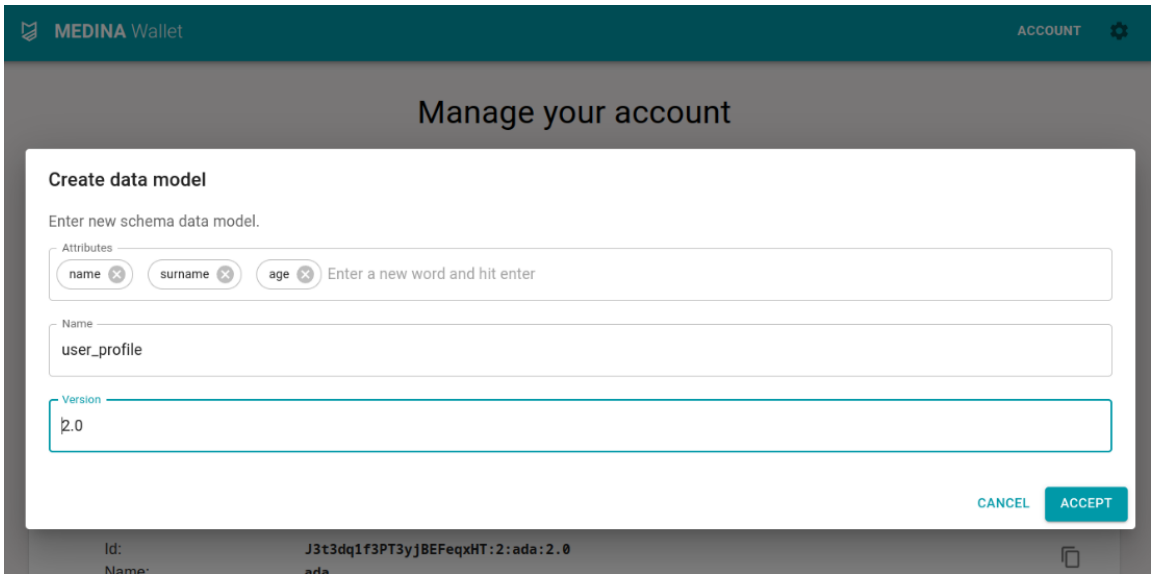

Figure 37. MEDINA SSI-webapp: Creation of a new data models

The "Owned schemas" tab allows a user to claim the ownership of a certain data model (see Figure 38) and list the owned schemas (see Figure 39).

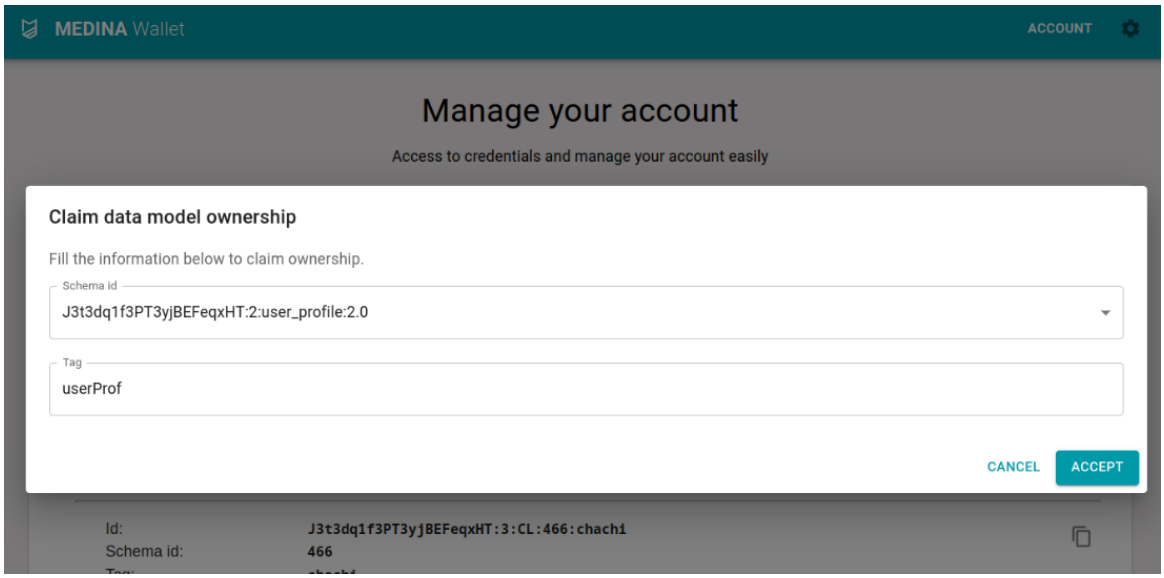

Figure 38. MEDINA SSI-webapp: Dialog which allows the user to claim the ownership of a data model

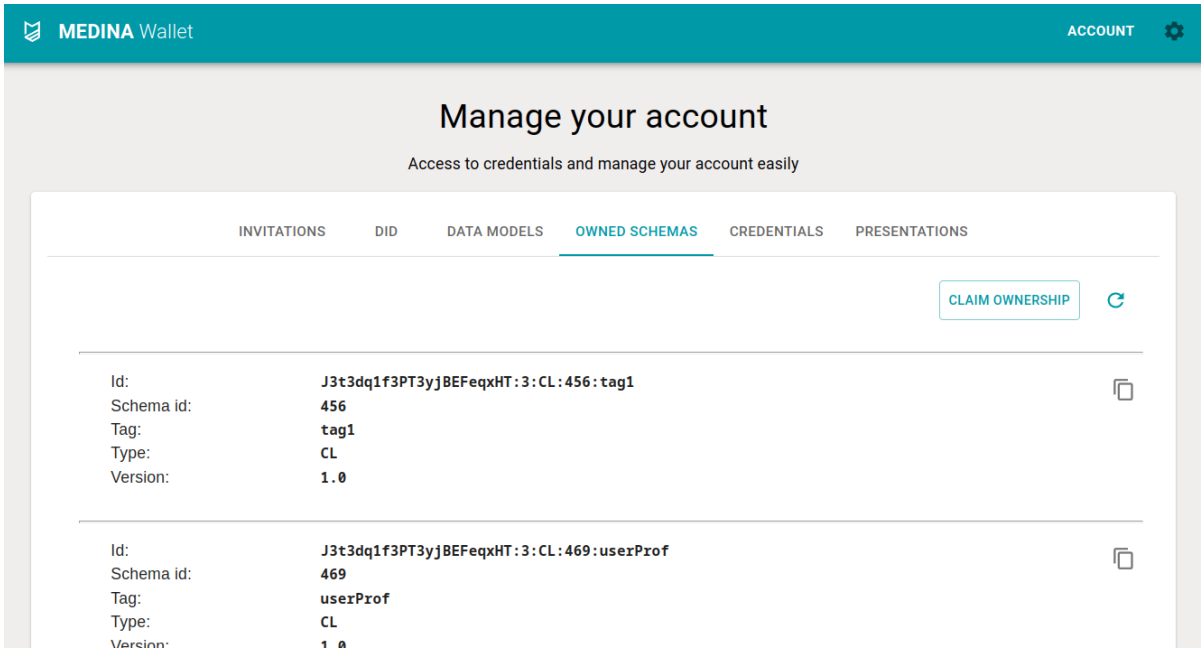

Figure 39. MEDINA SSI-webapp: "Owned schema" tab listing after claiming ownership of the "user_profile" schema

Issuing credentials

An issuer can create a new credential using the "Create credential" button in the "Credentials" tab. The dialog will allow to send a credential auto-offer to another SSI-agent. That is, the credential sent by the issuer will be automatically accepted by the receiver (holder) and added to its wallet.

To create a new credential, the user must select an active connection and an owned schema. After the schema selection, the form will be updated showing a text field for each of the attributes of this schema (see Figure 40).

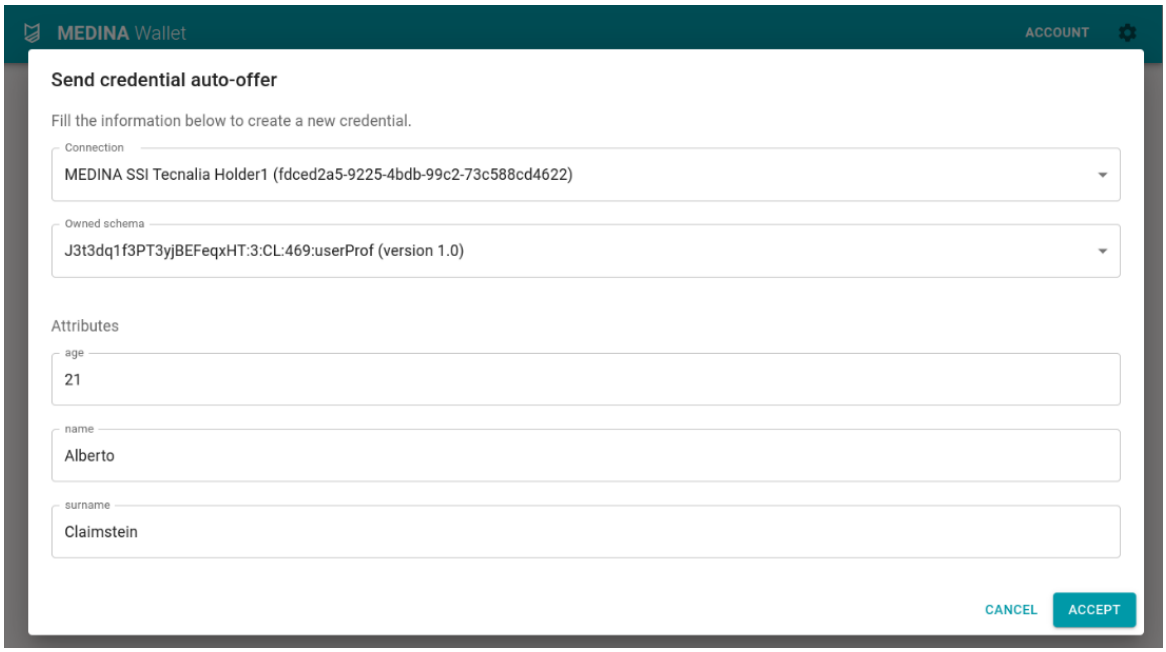

Figure 40. MEDINA SSI-webapp: Credential sending dialog with the credentials provided to "MEDINA SSI Tecnalia Holder1" for the "userProf" schema

The holder can then list all the owned credentials from different issuers (see Figure 41).

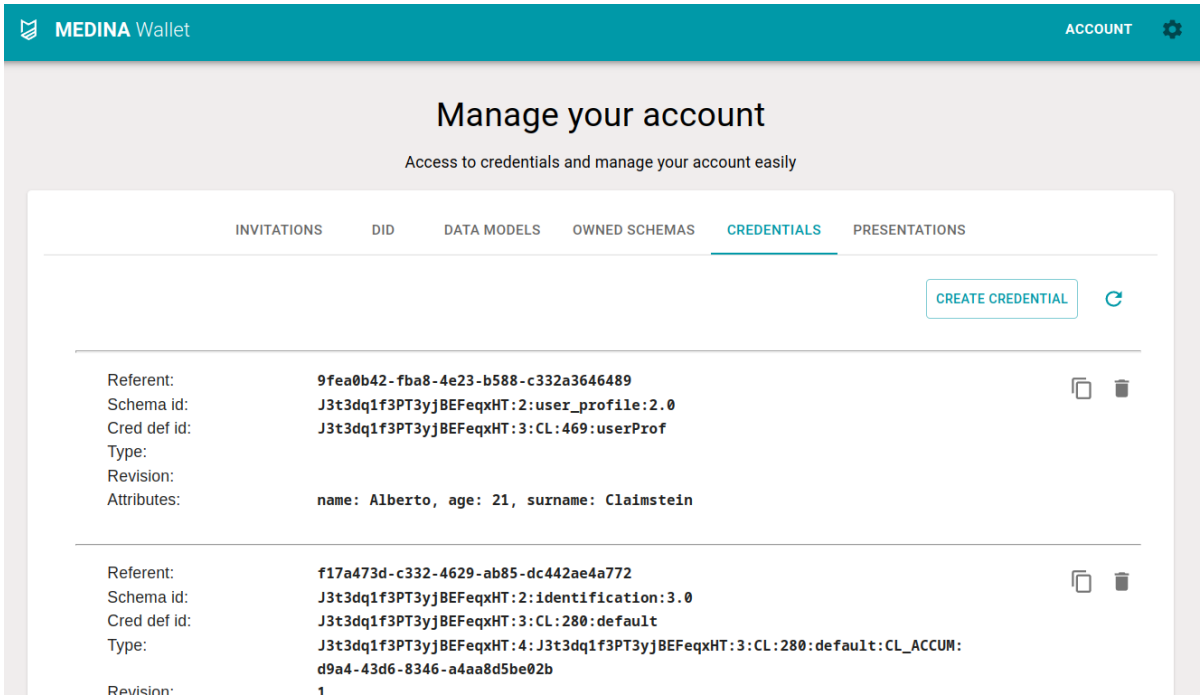

Figure 41. MEDINA SSI-webapp: "Credentials" tab showing the credentials of "MEDINA SSI Tecnalia Holder1"

Proof exchange

The "Presentations" tab shows the credentials presented to/by the current SSI agents. Any verifier can ask for a credential presentation using the "Request proofs" button in the "Presentations" tab (see Figure 42).

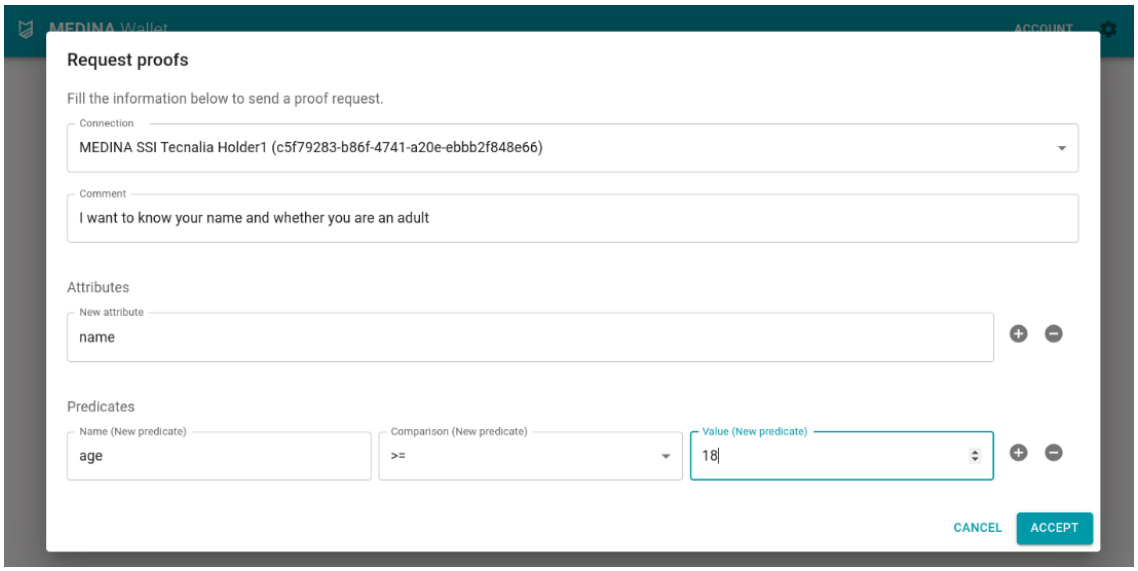

Figure 42. MEDINA SSI-webapp: Dialog used to claim a credential presentation

Once the verifier has requested some proofs, the prover account will see the new request listed (see Figure 43). Afterwards, the prover can click on the "reply" button to open a new dialog where the credential to be presented in the response can be selected (see Figure 44).

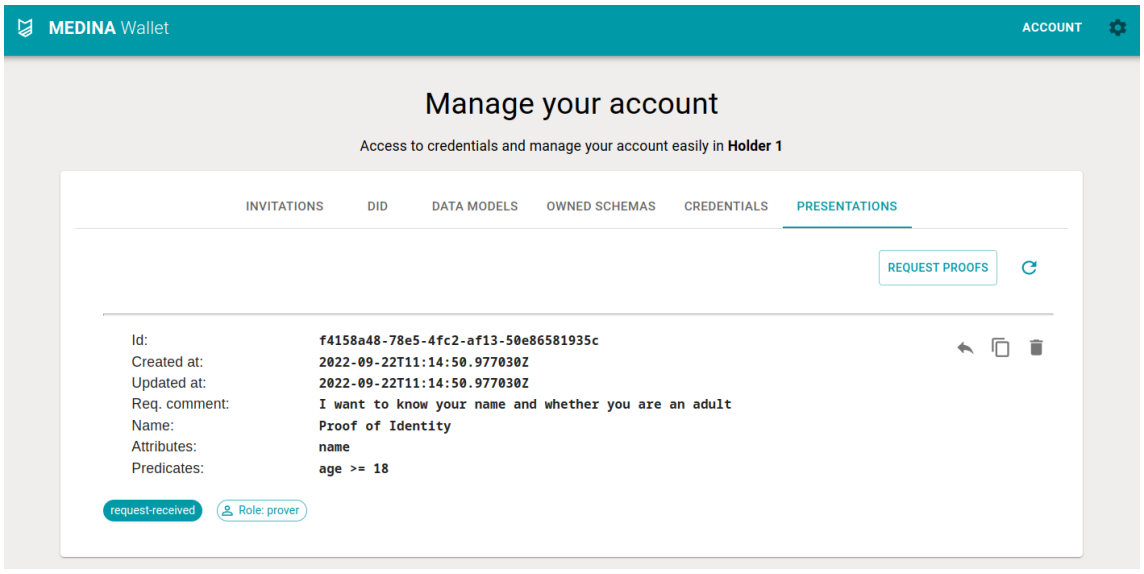

Figure 43. MEDINA SSI-webapp: Presentation tab seen by the prover account

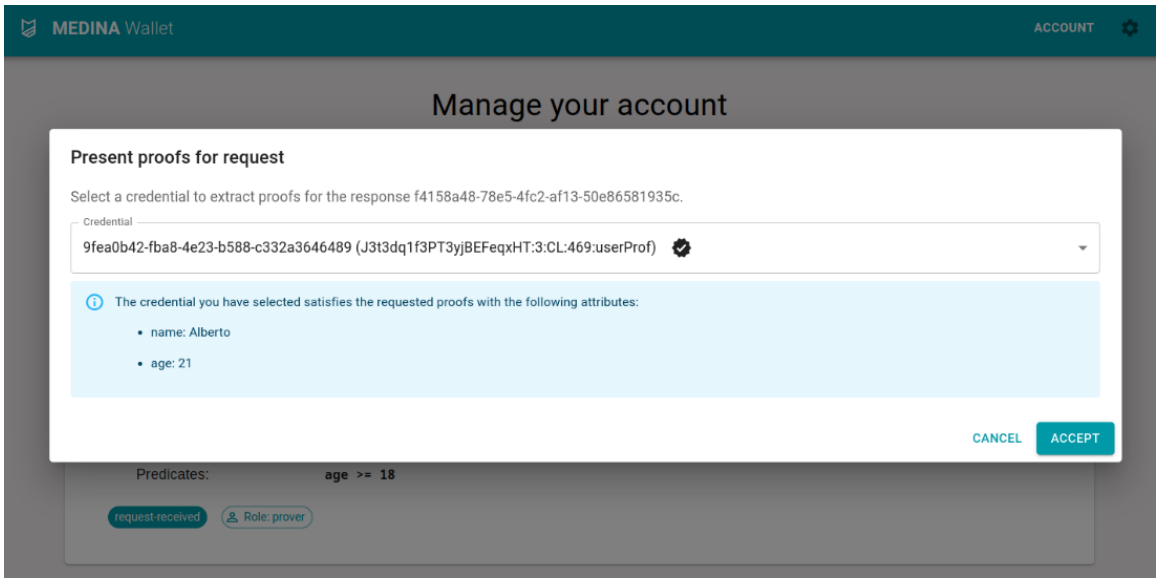

Figure 44. MEDINA SSI-webapp: Dialog used by a prover to manually choose the credential needed to answer to the presentation request

Additionally, the list of requested proofs can be checked (see Figure 45), where two proofs are shown: the proof requested in the previous screenshot and a proof presentation which has been abandoned because the prover did not present a credential which satisfies it.

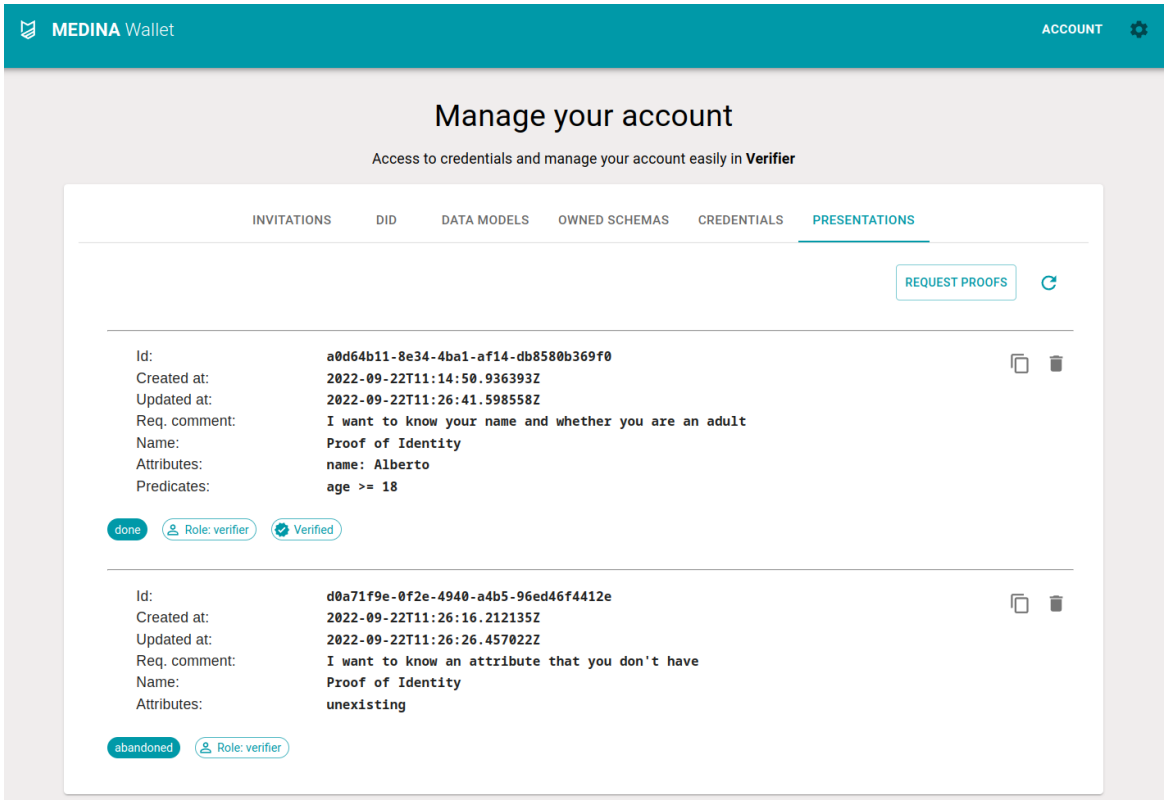

Figure 45. MEDINA SSI-webapp: "Presentations" tab showing the credentials presented to "MEDINA SSI Tecnalia Verifier"

6.3.3.4 Licensing information

6.3.3.5 Download

This section is not applicable in this current version as the whole components are provided as a service from TECNALIA for demo purposes and no download is needed by users.

In the next version, the holder instance will need to be downloaded and installed at the CSP premises as part of the MEDINA framework.

6.4 Risk Mitigation

Section 6.1 summarizes the identified risks that are associated with automatic certificate management. In the following we address each one briefly, describing the extent to which they are addressed by the design choices made in the *LCM* and *SSI Framework*.

1. **Modify the logic of the certificate management component:** While an attacker who modifies the *LCM* can change the certificate state (assuming its identifier is known), the manual verification is still in place.
2. **Forge a certificate:** A certificate can only be forged if the attacker obtains the CAB's private signing key
3. **Delete a certificate:** The *SSI Framework* stores certificates in a distributed way which is highly tamper-proof
4. **Deny the retrieval of a certificate:** The *SSI Framework* stores certificates in a distributed way which is highly available
5. **Disclose sensitive certificate details:** This risk will be further addressed in the final iteration.

6.5 Summary and Future Work

6.5.1 Summary

In this section, we have presented the second iteration of the *LCM*, as well as the SSI concept. The *LCM* has advanced primarily in the information it consumes and uses to make certificate maintenance decisions. It is now also better integrated with the *Orchestrator*, the CCE, and the *SSI Framework*. The *SSI Framework*, which was previously only described as a concept in D4.1 [4], is presented in this second iteration as a proof-of-concept implementation for demonstrating its functionality and the advantages of such a system in comparison to the state-of-the-art, like a Public Key Infrastructure (PKI).

6.5.2 Limitations and Future Work

Limitations of the *Life-Cycle Manager* include firstly that it focuses on risk value and operational effectiveness. This information may be too narrow; its usefulness would need to be shown in practical studies. Also, many changes in the certificates could be generated due to oscillating assessment results, which could overwhelm auditors. Finally, the security of certificates is limited by the general security measures that have been taken by a CSP, e.g., to secure the CSP's network, which is outside the scope of MEDINA.

Therefore, future work includes the following tasks:

- Regarding the *Life-Cycle Manager*:
  - Improve usability, e.g., preventing fast-changing state changes that could overwhelm auditors
  - Improve the integration with the *SSI Framework*: design a back-channel from the *SSI Framework* to accept/reject *LCM* decisions
  - Add more operational effectiveness metrics into the maintenance process

- Regarding the Self-Sovereign Identity System:
  - Define specific schemas for MEDINA
  - Improve usability of the SSI-webapp based on the use-cases feedback
  - Improve security of the SSI-webapp
  - Integrate the *SSI Framework* (holder) and the keycloak component
  - Improve privacy for the use-cases using ZKPs

7 Conclusions

The continuous certification of security properties in cloud services poses various challenges, including the continuous aggregation and evaluation of evidence, as well as the continuous management of certificates. Also, the protection of evidence integrity, i.e., its trustworthiness, is a major challenge, since it is essential to establish trust in the whole certification process.

This deliverable has presented the second iteration of concepts and prototypes for a *Continuous Certification Evaluation*, an Automated *Life-Cycle Manager*, a *Self-Sovereign Identity (SSI) Framework* for the issuance of certificates, as well as the concept for the trustworthiness of evidence and assessment results (whose implementation is described in WP3).

The technology evaluations in this deliverable have shown that some emerging technologies, like Blockchain and smart contracts, can provide benefits for the automation and protection of certificates. At the same time, they can introduce considerable overhead and new risks. Future work therefore must carefully balance practical considerations of CSPs with appropriate security and automation measures.

In the upcoming final iteration of this deliverable, the evaluation of assessment results will be improved, e.g., with further operational effectiveness metrics. Also, the current evaluations of distributed ledger technologies and smart contracts will be further extended to paint a more comprehensive picture of how those technologies can contribute to protecting the integrity of automated audit processes and artifacts focusing on energy consumption, cost, and scalability aspects. Further future improvements include improvements to the prototype implementations and their integrations into the overall MEDINA framework, and the testing in relation to the MEDINA use cases to get realistic feedback on the contributions of this work package.

8 References

[1] MEDINA Consortium, “D4.4 Methodology and tools for risk-based assessment and security control reconfiguration-v1,” 2021.

[2] MEDINA Consortium, “D2.1 Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1,” 2021.

[3] MEDINA Consortium, “D3.2 Tools and techniques for the management of trustworthy evidence-v2,” 2022.

[4] MEDINA Consortium, “D4.1 Tools and Techniques for the Management and Evaluation of Cloud Security Certifications - V1,” 2021.

[5] J. Luna, A. Taha, R. Trapero and N. Suri, "Quantitative Reasoning about Cloud Security Using Service Level Agreements," *IEEE Transactions on Cloud Computing,* vol. 5, no. 3, pp. 457-471, 2017.

[6] J. Luna, R. Langenberg and N. Suri, "Benchmarking cloud security level agreements using quantitative policy trees," in *CCSW '12: Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, Raleigh North Carolina USA, 2012.

[7] A. Taha, R. Trapero, J. Luna and N. Suri, "AHP-Based Quantitative Approach for Assessing and Comparing Cloud Security," in *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, 2014.

[8] J. Modic, R. Trapero, A. Taha, J. Luna, M. Stopar and N. Suri, "Novel efficient techniques for real-time cloud security assessment," *Computers & Security,* vol. 62, 2016.

[9] EU FP7 SPECS, "Secure Provisioning of Cloud Services based on SLA management," [Online]. Available: https://cordis.europa.eu/project/id/610795. [Accessed October 2022].

[10] S. Maroc and J. Biao Zhang, "Towards Security Effectiveness Evaluation for Cloud Services Selection following a Risk-Driven Approach," *International Journal of Advanced Computer Science and Applications (IJACSA),* vol. 11, no. 1, 2020.

[11] P. Stephanow and C. Banse, "Evaluating the performance of continuous test-based cloud service certification," in *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.

[12] ENISA, "EUCS – Cloud Services Scheme," [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme. [Accessed October 2022].

[13] S. Cimato, E. Damiani, F. Zavatarelli and R. Menicocci, "Towards the certification of cloud services," in *IEEE Ninth World Congress on Services*, Santa Clara, CA, USA, 2013.

[14] C. A. Ardagna, R. Asal, E. Damiani, N. El Ioini, C. Pahl and T. Dimitrakos, "A certification technique for cloud security adaptation," in *IEEE International Conference on Services Computing (SCC)*, San Francisco, CA, USA, 2016.

[15] I. Kunz and P. Stephanow, "A process model to support continuous certification of cloud services," in *IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, 2017.

[16] M. Anisetti, C. A. Ardagna, E. Damiani and F. Gaudenzi, "A semi-automatic and trustworthy scheme for continuous cloud service certification," *IEEE Transactions on Services Computing,* vol. 13, no. 1, pp. 30--43, 2017.

[17] AssureMoss Consortium, "D5.2. Methodology for Incremental and Continuous Certification Scheme of software," https://assuremoss.eu/en/resources/Deliverables/D5.2.-Methodology-for-Incremental-and-Continuous-Certification-Scheme-of-software, 2021.

[18] MEDINA Consortium, "D5.2 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy-v2," 2022.

[19] P. Torr, "Demystifying the threat modeling process," in *IEEE Security & Privacy*, 2005.

[20] R. M. Blank, "Guide for conducting risk assessments," in *Citeseer*, 2011.

[21] "Hyperledger Fabric," [Online]. Available: https://www.hyperledger.org/use/fabric. [Accessed October 2022].

[22] Consensys, "Quorum," [Online]. Available: https://github.com/ConsenSys/quorum. [Accessed October 2022].

[23] B. Preneel, "Cryptographic hash functions.," *European Transactions on Telecommunications,* vol. 5, no. 4, pp. 431-448, 1994.

[24] T. Baicheva, S. Dodunekov and P. Kazakov, "On the cyclic redundancy-check codes with 8-bit redundancy," *Computer Communications,* vol. 21, no. 11, pp. 1030-1033, 1998.

[25] M. Rjaško, "Properties of cryptographic hash functions," in *Cryptology ePrint Archive*, 2008.

[26] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International workshop on fast software encryption*, Heidelberg (Germany), 2004.

[27] A. E. d. P. d. Datos, "Introduction to the hash function as a personal data pseudonymisation technique," 20019. [Online]. Available: https://edps.europa.eu/data-protection/our-work/publications/papers/introduction-hash-function-personal-data_en. [Accessed October 2022].

[28] J. H. F. H. O. P. L. &. S. V. Horalek, "Analysis of the use of Rainbow Tables to break hash," *Journal of Intelligent & Fuzzy Systems,* vol. 32, no. 2, pp. 1523-1534, 2017.

[29] S. Nakamoto, "Bitcoin. A peer-to-peer electronic cash system.," *Decentralized Business Review,* no. 21260, 2008.

[30] L. Ante, "Smart contracts on the blockchain–a bibliometric analysis and review," *Telematics and Informatics,* 2020.

[31] M. Röscheisen, M. Baldonado, K. Chang, L. Gravano, S. Ketchpel and A. Paepcke, "The Stanford InfoBus and its service layers: Augmenting the Internet with higher-level information management protocols," *Digital Libraries in Computer Science: The MeDoc Approach,* pp. 213--230, 1998.

[32] W. Viriyasitavat, L. Da Xu, Z. Bi and A. Sapsomboon, "Blockchain-based business process management (BPM) framework for service composition in industry 4.0," *Journal of Intelligent Manufacturing,* vol. 31, no. 7, pp. 1737--1748, 2020.

[33] L. García-Bañuelos, A. Ponomarev, M. Dumas and I. Weber, "Optimized execution of business processes on blockchain," in *International conference on business process management*, 2017.

[34] B. Carminati, E. Ferrari and C. Rondanini, "Blockchain as a platform for secure inter-organizational business processes," in *Carminati, Barbara, Elena Ferrari, and Christian Rondanini. "Blockchain as a platform for secure inter-organizational business processes." 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, 2018.

[35] MEDINA Consortium, "D5.1 MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy-v1," 2021.

[36] V. Buterin, "Ethereum white paper. GitHub repository," 2013. [Online]. Available: https://ethereum.org/en/whitepaper/. [Accessed October 2022].

[37] M. Hearn, "Corda: A distributed ledger. Corda Technical White Paper," 2016.

[38] H. Sawtooth. [Online]. Available: https://www.hyperledger.org/use/sawtooth. [Accessed October 2022].

[39] "Hyperledger Besu explained," [Online]. Available: https://limechain.tech/blog/hyperledger-besu-explained/. [Accessed October 2022].

[40] "What is Amazon QLDB?," [Online]. Available: https://docs.aws.amazon.com/qldb/latest/developerguide/what-is.html. [Accessed October 2022].

[41] "About BigchainDB," [Online]. Available: https://www.bigchaindb.com/. [Accessed October 2022].

[42] "Tendermint," [Online]. Available: https://tendermint.com/. [Accessed October 2022].

[43] F. M. Schuhknecht, A. Sharma, J. Dittrich, Agrawal and Divya, "chainifyDB: How to get rid of your Blockchain and use your DBMS instead," *CIDR,* 2021.

[44] “CovenantSQL-The Blockchain SQL Database,” [Online]. Available: https://covenantsql.io/. [Accessed October 2022].

[45] “Fluree – The Web3 Data Platform,” [Online]. Available: https://flur.ee/. [Accessed October 2022].

[46] M. S. Sahoo and P. K. Baruah, "Hbasechaindb–a scalable blockchain framework on hadoop ecosystem," in *Asian Conference on Supercomputing Frontiers*, 2018.

[47] “What is HBase?,” [Online]. Available: https://www.ibm.com/topics/hbase. [Accessed October 2022].

[48] R. Sobti and G. Geetha, "Cryptographic hash functions: a review," *International Journal of Computer Science Issues (IJCSI),* vol. 9, no. 2, p. 461, 2012.

[49] F. Mendel, T. Nad and M. Schläffer, "Finding SHA-2 characteristics: searching through a minefield of contradictions," in *International Conference on the Theory and Application of Cryptology and Information Security*, Heidelberg (Germany), 2011.

[50] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," 2015.

[51] J. Czajkowski, L. Groot Bruinderink, A. Hülsing, C. Schaffner and D. Unruh, "Post-quantum security of the sponge construction," in *International Conference on Post-Quantum Cryptography*, 2018.

[52] C. Signing, "Hash Algorithm Comparison: MD5, SHA-1, SHA-2 & SHA-3," [Online]. Available: https://codesigningstore.com/hash-algorithm-comparison. [Accessed October 2022].

[53] X. Wang, D. Feng and X. Y. H. Lai, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD," *Cryptology EPrint Archive,* 2004.

[54] X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," in *Annual international conference on the theory and applications of cryptographic techniques*, 2005.

[55] H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," in *International Workshop on Fast Software Encryption*, Heidelberg (Germany), 1996.

[56] X. Wang, D. Feng and X. Yu, "An attack on hash function HAVAL-128.," *Science in China Series F: Information Sciences,* vol. 48, no. 5, pp. 545-556, 2005.

[57] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak and J. Szmidt, "Cryptanalysis of the GOST hash function," in *Annual International Cryptology Conference*, Heidelberg, 2008.

[58] D. C. Schmidt, "Gperf: A perfect hash function generator," 2000.

[59] Y. Yang, F. Shen, H. T. Shen, H. Li and X. Li, "Robust discrete spectral hashing for large-scale image semantic indexing," *IEEE Transactions on Big Data,* vol. 1, no. 4, pp. 162-171, 2015.

[60] S. Halevi, W. E. Hall and C. S. Jutla, "The Hash Function "Fugue"," *Cryptology ePrint Archive,* 2014.

Appendix A: Alternatives to Blockchain for Audit Trails

This section will theoretically compare the two existing alternatives to the use of Blockchain for Audit Trails, namely traditional databases or replicated databases, identifying the main advantages and disadvantages in each case.

1. Blockchain vs Traditional databases

At first glance, Blockchain and traditional databases can be considered similar, as both are used, broadly speaking, to store information in a distributed or centralized way. However, **Blockchain is more than just a database**. There are several differences between both technologies:

- **AUTHORITY:**
  - Blockchain: It is decentralized; no central control
  - Database: It is centralized; it is controlled by an administrator

In Blockchain, each node takes part in a consensus mechanism to check all transactions, with the same level of access and capability. In a traditional database, a central authority (administrator) controls the whole system. In this context, Blockchain takes advantage over traditional databases since trust in a central entity is not required.

- **ARCHITECTURE:**
  - Blockchain: Distributed
  - Database: Client-server architecture

In Blockchain, data is distributed among all nodes; each node stores a copy of the complete Blockchain so although some node is compromised, the rest can continue working. Therefore, single point of failure attacks are infeasible in Blockchain, gaining in robustness and fault tolerance over traditional databases where data is centrally stored in a server.

- **DATA HANDLING:**
  - Blockchain: Read and Write
  - Database: CRUD (Create, Read, Update and Delete)

Traditional databases provide additional functionalities over Blockchain (update and delete). One of the most important features is the ability to delete information. In Blockchain nothing can be deleted; any data included in the Blockchain will be recorded forever.

In the MEDINA context, it is not needed to update evidence and assessment results in the audit trail, as an update in any of them is considered new evidence or a new assessment result. In addition, being able to delete existing information is not a requirement for MEDINA.

- **INTEGRITY:**
  - Blockchain: Supported
  - Database: Malicious actors can modify database data

One of the most important features of Blockchain is integrity, which is achieved through the consensus mechanism in which all participating nodes check and validate all transactions, and the distribution of data between all nodes with each node storing a copy of the entire Blockchain, so it seems impossible for a malicious actor to modify the stored information or to include incorrect information in the Blockchain. In traditional databases, the system is vulnerable if the administrator is compromised; as it is a centralized party, it is more likely to happen.

- **TRANSPARENCY:**

  - Blockchain: Supported
  - Database: The administrator is who decides which data can be accessed

In Blockchain, every participating node has the same level of access and capability, creating a system in which transparency is guaranteed by design. On the contrary, in traditional databases, the administrator decides who can access the database and what actions can execute.

- **IMPLEMENTATION AND MAINTAINANCE COSTS:**
  - Blockchain: High
  - Database: Low

Blockchain is still a new technology, so it is still more expensive to implement and maintain than traditional databases based on "old" technologies. However, some of the existing Blockchain technologies are already widely deployed, and their costs are starting to come down.

- **PERFORMANCE:**
  - Blockchain: Low (due to the verification and consensus methods)
  - Database: Fast and with high scalability

Traditional databases are known for faster execution time and can handle millions of data at any given time. However, Blockchain is considerably slower because of carrying more operations, including signature verifications and consensus mechanisms. However, in the MEDINA context, a high performance in the audit trail is not a requirement.

2. Blockchain vs Replicated databases

Some of the traditional databases' main disadvantages can be improved by means of replication techniques, copying data from one database to another resulting in a distributed database system with all databases with the same level of information. However, Blockchain still differs from replicated databases in the following aspects:

- **REPLICATION:**
  - Blockchain: Transaction replication
  - Replicated Database: State replication

Blockchain replicates entire transactions so that its execution can be replayed by each participant node. Distributed databases, on the contrary, replicate the resulting log of read and write operations. For this purpose, a distributed database management system is needed to guarantee that updates, additions, and deletions performed on the data at any given database are automatically reflected in the data stored at all the other databases.

- **CONCURRENCY:**
  - Blockchain: No (serial execution)
  - Replicated Database: Yes

Most Blockchains support only serial execution as the transaction execution is not the real bottleneck in the Blockchain performance (the consensus mechanism usually is) and, by this way, the behaviour of smart contracts is deterministic when the transaction execution is replicated over many nodes, being easier to identify the ledger states. Distributed databases, on the contrary, employ sophisticated concurrency control mechanisms to extract as much concurrency as possible and improve performance.

Although concurrency is not a real concern in MEDINA, some recent Blockchains have started to adopt some simple concurrency techniques, such as, for example, in *Hyperledger Fabric* where transactions are executed in parallel against the ledger states before being sent for ordering.

Appendix B: Blockchain Technologies

This section analyses different Blockchain technologies in order to identify the one that best fits in the context of MEDINA.

1. Consensus Algorithms

First of all, it is not possible to compare different Blockchain technologies without introducing some of the most famous consensus algorithms. All technologies use their own consensus algorithm or a combination of some of them. Some of the most used consensus algorithms are described below.

Proof-of-Work (PoW): Proof-of-work-based consensus mechanisms require the resolution of a computationally expensive calculation to validate a block. Mining nodes (the nodes in the network responsible for validating or "mining" blocks) compete to solve the computation and mine the block and are rewarded with a fee. It was the first consensus mechanism used in Blockchain (used by *Bitcoin*), although alternatives have been emerging with the purpose of improving some of its aspects (energy consumption, risk of network centralization, etc.).

Proof-of-Stake (PoS): In this case, the node creating the block is selected deterministically. The richness (number of tokens accumulated by a node) of each node is positively involved in this selection. The main problem of this "nothing-at-stake" consensus algorithm is that when a chain is split, since it costs nothing, the two forks are bet on. This makes it so that consensus on a single Blockchain is not guaranteed. It is used by *Nxtcoin*, *Peercoin* or *Bitshares*, for example. Ethereum has decided to incorporate Proof of Stake by means of the Casper Protocol.

Proof of Authority (PoA): PoA is a modified form of PoS where instead of stake with the monetary value, a validator's identity performs the role of stake. In this context, identity means the correspondence between a validator's personal identification on the platform with officially issued documentation for the same person, i.e., certainty that a validator is exactly who that person represents to be. Just like in PoS, in PoA consensus, identity as a form of stake is also scarce. But unlike PoS, there's only one identity per person. *Kovan* and *Rinkeby*, the two Ethereum test nets, use PoA.

Casper protocol: Casper emerges as a hybrid between PoW and PoS and is currently the algorithm that Ethereum is trying to implement. That is why it is actually considered by some authors as a PoS type algorithm. Casper works as a kind of wager in which different nodes propose blocks that should be added to the chain. The validating nodes deposit an amount of currency (deposit) and receive a reward if they have behaved honestly and, on the contrary, they are penalized if they do not, losing their deposit. The nodes bet on the blocks that will be added and if the block turns out to be correct, they receive the reward, i.e., betting on the consensus implies winning coins, while betting against the consensus implies losing them. This system of incentives and penalties maintains the consistency of the network.

Proof-of-Elapsed Time (PoET): Each participant requests a timeout from their local trusted enclave. The participant with the shortest timeout is next to propose a block, after waiting the allotted timeout. Each local trusted enclave signs the function and the result so that other participants can verify that no one has cheated on the timeout.

Proof-of-Space (Proof-of-Capacity): In this case, the user "pays" with hard disk space. The more hard-disk space the user has, the better is the chance of extracting the next block and earning the block reward. The algorithm generates large data sets known as "plots", which must be stored on the users' hard disk. The more plots the user has, the better is the chance of finding

the next block on the chain. *Burstcoin* is the only cryptocurrency that currently uses a form of proof of capability.

Practical Byzantine Fault Tolerance (PBFT): This is a consensus algorithm that is normally used for consensus in distributed system but does not really meet the requirements for economic consensus on Blockchains since PBFT becomes infeasible in networks with a high number of nodes due to the required communication; Blockchain technologies using PBFT only rely on a reliable subnetwork of participants to establish consensus. Such a consensus algorithm is popular in private networks, being currently employed in *Hyperledger Fabric*, as it provides a way to reach consensus where the majority of nodes are assumed to be non-malicious.

Istanbul Byzantine Fault Tolerance (IBFT): IBFT is a variant of the PoA algorithm. Moving away from the more technical aspects of IBFT, the most important fact is that it is, along with Raft[5], one of the consensus algorithms employable in *Quorum* networks. In the same way as PBFT, it makes sense mainly in private Blockchain deployments.

2. Private vs Public

Blockchains can be public or private:

Public: A public Blockchain is open to the public and anyone can join without specific permission. All people who join the network can read, write, and participate in this network that is not controlled by anyone in particular.

Private: Private Blockchains are based on invitation and anyone who wants access to the Blockchain must ask for permission from the Blockchain's governing body. They allow different levels of access that determine which users can write, read, and audit the Blockchain. Thus, data is not public.

The main advantage of a private Blockchain is related to the control over the network participants, which is highly recommended in MEDINA, where the network should not be open to the public. In addition, the number of nodes needed to set up the network is limited, so the network is faster, more efficient, and more convenient in terms of time and energy consumption. This is mainly because the consensus in public Blockchains is more complex since it is necessary to protect the network from untrusted nodes, so extra verifications and operations must take place, while in private Blockchains it does not happen because the nodes which are in the network are under control; it logically takes more time to synchronize a network and reach consensus when more nodes are involved in the consensus process. Finally, private Blockchains can be free of charge, which is highly recommended for the MEDINA audit trail system.

3. Technical comparison

This section presents some of the most well-known Blockchain technologies with their main characteristics.

Bitcoin: Bitcoin [29] is a protocol conceived in 2008 by Satoshi Nakamoto, an anonymous person, that promises decentralized payments between parties with no central authority using peer-to-peer technology. Bitcoin promises to send value in form of tokens between different actors by paying a small amount of money as fee, offering the promise of lower transaction fees than traditional online payment mechanisms. There is no physical bitcoin, only balances kept on a public ledger that everyone has transparent access to.

[5] Raft is a CFT consensus algorithm

All bitcoin transactions are verified by a massive amount of computing power, which is commonly known as mining. Regarding their Blockchain characteristics, Bitcoin is public and permission less, as anyone has access to the shared ledger and can participate in the network.

Ethereum: Ethereum [36] includes open access to digital money and data-friendly services for everyone. It is a community-built technology behind its cryptocurrency ether (ETH). It also supports decentralized programmable Smart Contracts, which use ether to work. These Smart Contracts are implemented using Solidity language. Ethereum also has transaction fees, so users must pay a small amount of money to use the network, similarly to Bitcoin. It features a throughput of approximately 20 transactions per second, bettering Bitcoin. It is a public and permission less Blockchain.

Hyperledger Fabric: Hyperledger Fabric [21] is a platform for the implementation of distributed solutions. It is based on Blockchain, so it can take advantage of all the benefits provided by this technology. Fabric implements Smart Contracts using Go as programming language.

Hyperledger Fabric, unlike Bitcoin and Ethereum, is private which means that permissions are required for third parties to access the network, and it is also permissioned, so it is possible to set different permissions to different nodes in the network. This marks a profound difference with Ethereum when it comes to forming consensus, since in Ethereum the roles and tasks required to reach consensus are identical. In addition, due to its nature, it allows the implementation of private channels, so that it is possible to share information only with certain parties. Unlike Bitcoin or Ethereum, it does not have mining or its own token, so it is not possible to give it cryptocurrency functionalities. In addition, due to the smaller size of the networks, it does not present as many scalability problems as the previous ones.

Quorum: Quorum [22] is, according to the project page, an enterprise-focused version of Ethereum. The differences with Ethereum are therefore notable; on the one hand, it is permission-oriented and works on private networks. It also promises high speed and high performance, although logically it should not be compared with technologies such as Ethereum or Bitcoin, since, as they are focused on public networks, it is to be expected that performance and speed will be much lower due to a larger number of nodes.

Being based on Ethereum, it supports the use of Smart Contracts and has a token. Also, according to the project page, since it runs on Ethereum, it is easy to incorporate Ethereum functionalities into Quorum. As for the consensus algorithm, it uses PoS, although it can also work with other consensus algorithms.

Corda: Corda [37] is an open-source project based on Blockchain technology and designed to be used mainly by financial institutions. In terms of scalability, it has the same particularities as Hyperledger Fabric, as well as the consensus mechanism. However, Corda uses what are known as notary nodes, which provide evidence that a transaction has been carried out. This way of reaching consensus is state-based.

Like Hyperledger Fabric, it is private and permission-oriented and implements Smart Contracts, which can be mainly implemented in Java or Kotlin. Like Hyperledger Fabric, it does not have its own token.

Hyperledger Sawtooth: Hyperledger Sawtooth [38] is similar to Hyperledger Fabric, but in this case, it is designed to operate in IoT devices with little human interaction. It incorporates the consensus mechanism PoET. It has become well-known due to its ease of integration into security hardware solutions. In addition, it provides some advances over Hyperledger Fabric such as the ability to execute transactions in parallel and offers support for multiple languages and Ethereum. However, the project is still at a very early stage of development and, in addition,

having been developed by Intel, there are doubts about the range of hardware devices that will be able to work with this system.

Hyperledger Besu: Hyperledger Besu [39] is a java-based Ethereum client designed to be enterprise-friendly for both public and private permissioned network use cases. It can also be run on test networks such as *Rinkeby*, *Ropsten*, and *Görli*. Hyperledger Besu includes several consensus algorithms including PoW, and PoA (IBFT, IBFT 2.0, Etherhash, and Clique). Its comprehensive permissioning schemes are designed specifically for use in a consortium environment. The project, formerly known as Pantheon, joined the Hyperledger family in 2019, adding for the first time a public blockchain implementation to Hyperledger's suite of private blockchain frameworks. Whereas Hyperledger Fabric is a private protocol designed from the ground up to support enterprise-grade solutions, Besu seeks to utilize the public Ethereum network.

Regarding Hyperledger Besu, the Besu client is designed to be highly modular to ensure that key Blockchain features such as consensus algorithms can be easily implemented and upgraded. The goal here is to provide businesses with the means to easily configure Ethereum according to their needs while enabling smooth integration with other Hyperledger projects, such as Hyperledger Fabric.

Its smart approach of using the Ethereum Blockchain affords developers enough flexibility to build public or permissioned solutions based on the specific requirements of each use case. Rather than a comparison between Hyperledger Besu and Hyperledger Fabric, it is important to remark than both technologies are complementary and solve different problems. However, Hyperledger Besu presents advantages against Hyperledger Fabric in terms of interoperability, because it can be integrated as an enterprise client in any Ethereum network. It also has compatibility with Quorum. Hence, it fulfils more integration requirements than Hyperledger Fabric, which can only use its own network. Also, due to its compatibility with Ethereum, Hyperledger Besu allows the use of tokens.

Amazon QLDB: Amazon Quantum Ledger Database (Amazon QLDB) [40] is a fully managed ledger database that provides a transparent, immutable, and cryptographically verifiable transaction log owned by a central trusted authority. Amazon QLDB can be used to track all application data changes and maintain a complete and verifiable history of changes over time. Amazon QLDB is a new class of database that helps eliminate the need to engage in the complex development effort of building your own ledger-like applications. With QLDB, the history of changes to your data is immutable.

Amazon QLDB works as a "Blockchain-as-a-Service" (BaaS) where Amazon provides the infrastructure. One of the main drawbacks is that governance is fully managed by Amazon and data is stored on Amazon´s side, centralizing the storage in a single provider. Also, it has associated costs as it works as-a-Service.

BigchainDB: BigchainDB [41] was mainly developed to combine the best characteristics of the "traditional" distributed database and the "traditional" Blockchain. It uses MongoDB as database and allows queries over the stored data, while preserving the immutability and decentralization; and Tendermint [42] as Blockchain framework. It has low latency and presents a better throughput than other Blockchains, such as Bitcoin and Ethereum. However, this is not a fair comparison as it is permissioned and uses BFT algorithm to reach consensus, which is significantly faster than other algorithms used in public networks. One of its strong points is that it can be easily integrated in traditional stacks, providing a decentralized and immutable ledger where data and transactions can be stored.

ChainifyDB: ChainifyDB [43] presents itself as a solution to integrate existing databases with Blockchain technology. It proposes the installation of a lightweight Blockchain layer on top.

ChainifyDB is a permissioned Blockchain layer which can be integrated into an existing heterogeneous database landscape adding a low overhead (8.5%) on the underlying database systems. It also promises up to 6x higher throughput than Hyperledger Fabric.

CovenantSQL: CovenantSQL [44] is a BFT relational database built on a standard SQLite, powered by a decentralized query engine. Hence, it seems to work as a private and permissioned Blockchain. It is an open-source alternative of Amazon QLDB. It also achieves decentralization by using peer-to-peer technology and keeps the integrity of the data stored in it. At the current date, they are still working on the whitepaper.

FlureeDB: FlureeDB [45] is an enterprise Blockchain-based database solution that combines Blockchain's security, immutability, decentralization and distributed ledger capabilities with a feature-rich graph-style database. It is composed by a database and a permissioned Blockchain. Regarding the ledger, it can be kept private among a consortium of entities or public for everyone.

FlureeDB deviates from other Blockchain technologies, such as Hyperledger Fabric or Ethereum, by focusing on queries and being optimized for read performance. Hence, it can be used as a complement to these technologies, rather than a direct rival, by for example storing transactions´ data.

HBasechainDB: HBasechainDB [46] is a big data storage system for distributed computing based on Blockchain. It achieves immutability and decentralization thanks to Blockchain and uses a HBase database. An HBase database [47] is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS) and is fault-tolerant. This database is not compatible with structured query languages, such as SQL, so it clearly deviates from other alternatives, such as CovenantSQL or ChainifyDB. Its scope seems therefore quite limited to big data applications, in particular those running Hadoop. Finally, HBasechainDB is permissioned, as only authorised nodes are able to submit transactions.

In practice, HBasechainDB follows a similar approach than BigchainDB, but it uses Hadoop database instead of MongoDB. However, HBasechainDB seems to be more appropriate for big data applications as it uses Hadoop database.

Although some of the more recent aforementioned technologies present some advantages in terms of performance, they also present some concerns in terms of governance because the network is under the control of an enterprise. In addition, most of developers are currently more familiar with more traditional Blockchain technologies, such as Ethereum or Hyperledger Fabric, which also have a big community behind them. Hence, they are much more appropriate in terms of support and compatibility for the audit trail in MEDINA.

Appendix C: Current Leading Hash Algorithms

Various types of hash functions have been developed in the past [48]. Some important ones are described below:

SHA-2 [49]

The SHA algorithm (Secure Hash Algorithm) was originally created by the NSA and NIST with the aim of generating unique hashes or codes based on a standard. In 1993 the first SHA protocol, also called SHA-0, was born, but it was hardly used and did not have much impact. A couple of years later, an improved, more robust, and secure variant, SHA-1, was released, which has been used for many years to sign SSL/TLS digital certificates for millions of websites. A few years later SHA-2 was created, which has four variants depending on the number of output bits, namely SHA2-224, SHA2-256, SHA2-384 and SHA2-512. Currently, for security reasons, SHA-1 is no longer used, but it is highly recommended to use SHA2 or SHA3 (within the SHA family).

Among the many ways to create hashes, the SHA2-256 algorithm is one of the most used thanks to its balance between security and speed, it is a very efficient algorithm and has a high resistance to collisions. For example, the method of verifying Bitcoins is based on SHA2-256. The main characteristics for the different types of SHA-2 are:

- **Output size:** The size of characters that will form the hash.
- **Internal state size:** The internal hash sum, after each compression of a data block.
- **Block size:** The size of the block handled by the algorithm.
- **Maximum message size:** The maximum size of the message on which the algorithm is applied.
- **Word length:** The length in bits of the operation applied in each round by the algorithm.
- **Interactions or rounds:** The number of operations performed by the algorithm to obtain the hash.
- **Supported operations:** The operations performed by the algorithm to obtain the hash.

SHA-256

It has an output size of 256 bits, an internal state size of 256 bits, a block size of 512 bits, the maximum message size it can handle is 264 - 1, the word length is 32 bits, and the number of rounds is 64, as well as the operations it applies to the hash are +, and, or, xor, shr and rot.

SHA2-384

This algorithm is different in terms of features, but its operation is the same. It has an output size of 384 bits, an internal state size of 512 bits, a block size of 1024 bits, the maximum message size it can handle is 2128 - 1, the word length is 64 bits, and the number of rounds is 80, as well as the operations it applies to the hash are +, and, or, xor, shr and rot. This algorithm is a more secure version than SHA2-256, since more rounds of operations are applied, and it can also be applied on more extensive information. This hash algorithm is often used to check message integrity and authenticity in virtual private networks. On the downside, it is slower than SHA2-256, but in certain circumstances it can be suitable.

SHA2-512

As in all SHA-2, the operation is the same, changing only one feature. It has an output size of 512 bits. The rest of the features are the same as SHA2-384. 512 bits of internal state size, 1024 bits of block size, 2128 - 1 for the maximum message size, 64 bits of word length, and 80 is the

number of rounds. This algorithm also applies the same operations on each round +, and, and, or, xor, shr and rot.

SHA2-224

SHA2-224 has not mentioned as the main one, because its big brother (SHA2-256) is much more widely used, since the computational difference between the two is negligible and SHA2-256 is much more standardized. No collisions have been found for this algorithm, which makes it a safe and usable option.

SHA-3 [50]

SHA-3 is the most recent hash algorithm belonging to the SHA family; it was released by the NIST in 2015, but it is not yet being widely used. Although it is part of the same family, its internal structure is quite different. This new hashing algorithm is based on sponge construction [51]. This sponge construction is based on a random function or random permutation of data; it allows any amount of data to be input and any amount of data to be generated: the data is "absorbed" and processed to display an output with the desired length. In the data absorption phase, the XOR operation is used and then transformed into a permutation function. SHA-3 allows additional bits of information, to protect from extension attacks, something that happens with SHA-1 or even with SHA-2.

Another important feature is that it is very flexible, making it possible to test cryptanalytic attacks and use it in lightweight applications. Currently SHA2-512 is twice as fast as SHA3-512, but the latter could be implemented through hardware, in which case it could be even faster.

SHA-3 was born as an alternative to SHA-2, but not because using SHA-2 is insecure, but a plan B was considered necessary in case of a successful attack against SHA-2. In this way, both SHA-2 and SHA-3 will coexist for many years. Its final goal is to replace SHA-2 in typical TLS or VPN protocols that use this hashing algorithm to check data integrity and data authenticity.

Although SHA-2 and SHA-3 have been proven to be the most secure hash functions today with a good trade-off between security and performance, **SHA-2 is considered even more secure** as it can be shown in Table 14 [52]. SHA-3 is usually considered when there is a specific problem with SHA-2.

Table 14. SHA-2 and SHA-3 comparison

	SHA-2	SHA-3
Possibility of Collision	No proof of collision has been found yet.	Susceptible to collision in squeeze attack.
Weakness	• SHA 256 is slower than its previous versions. • Software and browsers must be updated to implement SHA2.	Susceptible to collision
In use?	Yes	Yes
Applications	• Security application protocols. • Cryptographic transactions. • Digital certificates.	Can replace SHA2 where necessary.

MD5 [53]

The MD (Message Digest) family was created in 1974 by Ron Rivest, a cryptographer and professor at MIT. MD2 was the first hashing system he created, focused on 8-bit computers, so it is easy to deduce that it has suffered numerous attacks and cannot be considered secure.

MD4 [54] was considered insecure because its hash calculation was not sufficiently complex. Although MD4 hashes resemble MD5 hashes, in MD5 there are many more steps added to the calculation to increase the complexity. MD5 was quite secure for many years, but today it is no longer of sufficient complexity for cryptographic and data encryption purposes. Computers have become powerful enough to crack MD5 hashes easily, so its use is currently limited.

RIPEMD-160 [55]

RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest) is a 160-bit message digest algorithm developed in Europe by Hans Dobbertin, Antoon Bosselaers and Bart Preneel, and first published in 1996. It is an improved version of RIPEMD, which was based on the design principles of the MD4 algorithm and is similar in security and performance to the more popular SHA-1.

There are also 128-bit, 256-bit and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256 and RIPEMD-320 respectively. The 128-bit version was intended only as a replacement for the original RIPEMD, which were also 128-bit and had some security concerns. The 256-bit and 320-bit versions only decrease the possibility of accidental hash collisions, and do not have higher levels of security than RIPEMD-128 and RIPEMD-160.

RIPEMD-160 was designed in the open academic community, in contrast to the SHA-1 algorithm, designed by the US National Security Agency (NSA). RIPEMD-160 is a less popular design and correspondingly less well studied than SHA functions.

Other hash functions

Other hash functions that are not so widely used today but have been important throughout history include: HAVAL [56], GOST [57], Gperf [58], Spectral [59] or Fugue [60], among others.